# **Building the Software Distribution**

* [Building on all systems with CMake](#1ksv4uv).
* [Building on a UNIX system with Autoconf](#3dy6vkm).
* [Building on an MS-DOS or Windows system with nmake](#17dp8vu).
* [Building on a VMS system](#3rdcrjn).
* [Building the Software on Other Systems.](#26in1rg)

This chapter contains step-by-step instructions on how to configure and build the TIFF software distribution. The software is most easily built on a UNIX system, but with a little bit of work it can easily be built and used on other non-UNIX platforms.

## Building on all systems with CMake

CMake may be used to generate build files for most common build systems and IDEs, and supports all UNIX-like systems as well as Windows. See the [CMake website](http://www.cmake.org/) for further details. To build the software on you need to first run cmake to configure the build and generate the system-specific build files. This reads the top-level CMakeLists.txt file, which probes the target system for necessary tools and functions, checks any options you specified to configure the build, and then outputs build files configured for your system. If using Unix Makefiles, once configuration is done, you simply run make (or gmake) to build the software and then make install to do the installation. For other build systems, you do the equivalent steps with the tool for that system. For example, on any UNIX system:

% **cd ./tiff-4.0.5**  
% **cmake**  
 *...lots of messages...*  
% **make**  
 *...lots of messages...*  
% **make test**  
 *...lots of messages...*  
# **make install**

Building is dependent on a make utility and a C (and optionally a C++ compiler), so you will need these tools.

In general, the software is designed such that the following targets will always be available

make [all] build stuff  
make test run the test suite  
make install build and install stuff  
make clean remove object files, executables and cruft

### Build Trees

There are two schemes for configuring and building the software. If you intend to build the software for only one target system, you can configure the software so that it is built in the same directories as the source code.

% **gzip -dc tiff-4.0.5.tar.gz | tar -xf -**  
% **cd ./tiff-4.0.5**  
% **cmake**  
% **make**  
% **make test**  
% **make install**

Otherwise, you can configure a build tree that is parallel to the source tree hierarchy (or in some completely different place) but which contains only configured files and files created during the build procedure.

% **gzip -dc tiff-4.0.5.tar.gz | tar -xf -**  
% **mkdir tiff-4.0.5-build**  
% **cd ./tiff-4.0.5-build**  
% **cmake ../tiff-4.0.5**  
% **make**  
% **make test**  
% **make install**

This second scheme is useful for:

* building multiple targets from a single source tree
* building from a read-only source tree (e.g. if you receive the distribution on CD-ROM)
* sharing the source files via a network, but building on multiple systems
* keeping the source tree clean (unlike autoconf, cmake does not provide a distclean target, so out of source builds are recommended)

### Generators

The default generator for UNIX is Unix Makefiles, and on Windows is NMake Makefiles or MSBuild depending upon the setup. Run **cmake --help** to list all the generators available for your platform. For example, to use the Ninja [build system](https://martine.github.io/ninja/) on UNIX or Windows:

**cmake -G Ninja**  
**cmake --build .**  
**ctest -V**  
**cmake --build . --target install**

Note that **cmake --build .** is a build-system-independent way of building a target; you can always use the build system directly.

Alternatively, using the MSBuild system on Windows (64-bit Release build with VS2013):

**cmake -G "Visual Studio 12 2013 Win64"**  
**cmake --build . --config Release**  
**ctest -V -C Release**  
**cmake --build . --config Release --target install**

With the above configuration, it's also possible to open the generated solution file with the Visual Studio IDE as well as building on the command-line.

### Configuration Options

The configuration process is critical to the proper compilation, installation, and operation of the software. The CMakeLists.txt script runs a series of tests to decide whether or not the target system supports required functionality and, if it does not, whether it can emulate or workaround the missing functions. After running cmake, check the CMakeCache.txt file; this contains all the results of the checks performed and the options set by the user. If cmake failed to run, check CMakeFiles/CMakeOutput.log and CMakeFiles/CMakeError.log; these should record the error which caused the failure.

A second function of the configure script is to set the default configuration parameters for the software. Of particular note are the directories where the software is to be installed. By default the software is installed in the **/usr/local** hierarchy. To change this behaviour the appropriate parameters can be specified on the command line. Run **cmake --help** to get a full list of possible options, and **cmake -LH** to list all the configurable options for this software package, or **cmake -LAH** to show all advanced options in addition. Standard installation related options are shown below.

Installation directories:  
 CMAKE\_INSTALL\_PREFIX  
  
Fine tuning of the installation directories:  
 CMAKE\_INSTALL\_BINDIR user executables [PREFIX/bin]  
 CMAKE\_INSTALL\_SBINDIR system admin executables [PREFIX/sbin]  
 CMAKE\_INSTALL\_LIBEXECDIR program executables [PREFIX/libexec]  
 CMAKE\_INSTALL\_SYSCONFDIR read-only single-machine data [PREFIX/etc]  
 CMAKE\_INSTALL\_SHAREDSTATEDIR modifiable architecture-independent data [PREFIX/com]  
 CMAKE\_INSTALL\_LOCALSTATEDIR modifiable single-machine data [PREFIX/var]  
 CMAKE\_INSTALL\_LIBDIR object code libraries [PREFIX/lib]  
 CMAKE\_INSTALL\_INCLUDEDIR C header files [PREFIX/include]  
 CMAKE\_INSTALL\_OLDINCLUDEDIR C header files for non-gcc [/usr/include]  
 CMAKE\_INSTALL\_DATAROOTDIR read-only arch.-independent data root [PREFIX/share]  
 CMAKE\_INSTALL\_DATADIR read-only architecture-independent data [DATAROOTDIR]  
 CMAKE\_INSTALL\_LOCALEDIR locale-dependent data [DATAROOTDIR/locale]  
 CMAKE\_INSTALL\_MANDIR man documentation [DATAROOTDIR/man]  
 CMAKE\_INSTALL\_DOCDIR documentation root [DATAROOTDIR/doc/tiff]

Also see the CMake [documentation](http://www.cmake.org/cmake/help/v3.3/) for [additional variables](http://www.cmake.org/cmake/help/v3.3/manual/cmake-variables.7.html) which may be set.

### Configuring Optional Packages/Support

The TIFF software comes with several packages that are installed only as needed, or only if specifically configured at the time the configure script is run. Packages can be configured via the **cmake** commandline parameters. *Static/Shared Objects Support* BUILD\_SHARED\_LIBS[=ON|OFF]    build shared libraries [default=ON]

This option controls whether or not to configure the software to build a shared and static binaries for the TIFF library. Use of shared libraries can significantly reduce the disk space needed for users of the TIFF software. If shared libraries are not used then the code is statically linked into each application that uses it.

ld-version-script[=ON|OFF]  Enable linker version script (default is ON)

Add shared library symbol versioning on ELF-based systems (e.g. Linux and FreeBSD) which use the GNU linker. This is needed if several major versions of libtiff might be loaded at once into the same program.

*JPEG Support* jpeg[=ON|OFF]        enable IJG JPEG library usage (required for JPEG compression, enabled by default)

JPEG\_INCLUDE\_DIR=DIR location of IJG JPEG library headers

JPEG\_LIBRARY=DIR     location of IJG JPEG library binary) The JPEG package enables support for the handling of TIFF images with JPEG-encoded data. Support for JPEG-encoded data requires the Independent JPEG Group (IJG) libjpeg distribution; this software is available at <http://www.ijg.org/>. **cmake** script automatically tries to search for a working IJG JPEG installation. If it fails to find library, JPEG support will be automatically disabled. If you want specify the exact paths to library binary and headers, use above options for that. *ZIP Support* The ZIP support enables support for the handling of TIFF images with deflate-encoded data (enabled by default if available). Support for deflate-encoded data requires the freely available zlib distribution written by Jean-loup Gailly and Mark Adler; this software is available at <http://www.zlib.org/>.

## Building on a UNIX System with Autoconf

To build the software on a UNIX system you need to first run the configure shell script that is located in the top level of the source directory. This script probes the target system for necessary tools and functions and constructs a build environment in which the software may be compiled. Once configuration is done, you simply run make (or gmake) to build the software and then make install to do the installation; for example:

% **cd ./tiff-4.0.5**  
% **./configure**  
 *...lots of messages...*  
% **make**  
 *...lots of messages...*  
% **make check**  
 *...lots of messages...*  
# **make install**

Supplied makefiles are dependent on a make utility and a C (and optionally a C++ compiler), so you will need these tools.

In general, the software is designed such that the following should be ``*make-able*'' in each directory:

make [all] build stuff  
make check run the test suite  
make install build and install stuff  
make clean remove object files, executables and cruft  
make distclean remove everything, that can be recreated

Note that after running "make distclean" the configure script must be run again to create the Makefiles and other make-related files.

### Build Trees

There are two schemes for configuring and building the software. If you intend to build the software for only one target system, you can configure the software so that it is built in the same directories as the source code.

% **gzip -dc tiff-4.0.5.tar.gz | tar -xf -**  
% **cd ./tiff-4.0.5**  
% **./configure**  
% **make**  
% **make check**  
% **make install**

Otherwise, you can configure a build tree that is parallel to the source tree hierarchy (or in some completely different place) but which contains only configured files and files created during the build procedure.

% **gzip -dc tiff-4.0.5.tar.gz | tar -xf -**  
% **mkdir tiff-4.0.5-build**  
% **cd ./tiff-4.0.5-build**  
% **../tiff-4.0.5/configure**  
% **make**  
% **make check**  
% **make install**

This second scheme is useful for:

* building multiple targets from a single source tree
* building from a read-only source tree (e.g. if you receive the distribution on CD-ROM)
* sharing the source files via a network, but building on multiple systems

### Configuration Options

The configuration process is critical to the proper compilation, installation, and operation of the software. The configure script runs a series of tests to decide whether or not the target system supports required functionality and, if it does not, whether it can emulate or workaround the missing functions. This procedure is fairly complicated and, due to the nonstandard nature of most UNIX systems, prone to error. The first time that you configure the software for use you should check the output from the configure script and look for anything that does not make sense for your system.

A second function of the configure script is to set the default configuration parameters for the software. Of particular note are the directories where the software is to be installed. By default the software is installed in the **/usr/local** hierarchy. To change this behaviour the appropriate parameters can be specified on the command line to configure. Run **./configure --help** to get a full list of possible options. Standard installation related options are shown below.

Installation directories:  
 --prefix=PREFIX install architecture-independent files in PREFIX  
 [/usr/local]  
 --exec-prefix=EPREFIX install architecture-dependent files in EPREFIX  
 [PREFIX]  
  
By default, `make install' will install all the files in  
`/usr/local/bin', `/usr/local/lib' etc. You can specify  
an installation prefix other than `/usr/local' using `--prefix',  
for instance `--prefix=$HOME'.  
  
For better control, use the options below.  
  
Fine tuning of the installation directories:  
 --bindir=DIR user executables [EPREFIX/bin]  
 --sbindir=DIR system admin executables [EPREFIX/sbin]  
 --libexecdir=DIR program executables [EPREFIX/libexec]  
 --sysconfdir=DIR read-only single-machine data [PREFIX/etc]  
 --sharedstatedir=DIR modifiable architecture-independent data [PREFIX/com]  
 --localstatedir=DIR modifiable single-machine data [PREFIX/var]  
 --libdir=DIR object code libraries [EPREFIX/lib]  
 --includedir=DIR C header files [PREFIX/include]  
 --oldincludedir=DIR C header files for non-gcc [/usr/include]  
 --datarootdir=DIR read-only arch.-independent data root [PREFIX/share]  
 --datadir=DIR read-only architecture-independent data [DATAROOTDIR]  
 --localedir=DIR locale-dependent data [DATAROOTDIR/locale]  
 --mandir=DIR man documentation [DATAROOTDIR/man]  
 --docdir=DIR documentation root [DATAROOTDIR/doc/tiff]  
 --htmldir=DIR html documentation [DOCDIR]  
  
Program names:  
 --program-prefix=PREFIX prepend PREFIX to installed program names  
 --program-suffix=SUFFIX append SUFFIX to installed program names  
 --program-transform-name=PROGRAM run sed PROGRAM on installed program names

### Configuring Optional Packages/Support

The TIFF software comes with several packages that are installed only as needed, or only if specifically configured at the time the configure script is run. Packages can be configured via the **configure** script commandline parameters. *Static/Shared Objects Support* --enable-shared[=PKGS]    build shared libraries [default=yes]

--enable-static[=PKGS]    build static libraries [default=yes]

These options control whether or not to configure the software to build a shared and static binaries for the TIFF library. Use of shared libraries can significantly reduce the disk space needed for users of the TIFF software. If shared libraries are not used then the code is statically linked into each application that uses it. By default both types of binaries is configured.

--enable-rpath             Enable runtime linker paths (-R libtool option)

Add library directories (see other options below) to the TIFF library run-time linker path.

--enable-ld-version-script  Enable linker version script (default is disabled)

Add shared library symbol versioning on ELF-based systems (e.g. Linux and FreeBSD) which use the GNU linker. This is needed if several major versions of libtiff might be loaded at once into the same program.

*JPEG Support* --disable-jpeg    disable IJG JPEG library usage (required for JPEG compression, enabled by default) --with-jpeg-include-dir=DIR    location of IJG JPEG library headers --with-jpeg-lib-dir=DIR    location of IJG JPEG library binary) The JPEG package enables support for the handling of TIFF images with JPEG-encoded data. Support for JPEG-encoded data requires the Independent JPEG Group (IJG) libjpeg distribution; this software is available at <http://www.ijg.org/>. **configure** script automatically tries to search for a working IJG JPEG installation. If it fails to find library, JPEG support will be automatically disabled. If you want specify the exact paths to library binary and headers, use above switches for that. *ZIP Support* The ZIP support enables support for the handling of TIFF images with deflate-encoded data. Support for deflate-encoded data requires the freely available zlib distribution written by Jean-loup Gailly and Mark Adler; this software is available at <http://www.zlib.org/>. If ZIP support is enabled the DIRS\_LIBINC and DIR\_GZLIB parameters should also be set (see below). By default this package is not configured.

## Building the Software under Windows 2000/XP/7/8/10 with nmake

With Microsoft Visual C++ installed, and properly configured for commandline use (you will likely need to source VCVARS32.BAT in AUTOEXEC.bAT or somewhere similar) you should be able to use the provided makefile.vc.

The source package is delivered using Unix line termination conventions, which work with MSVC but do not work with Windows 'notepad'. If you use unzip from the [Info-Zip](http://www.info-zip.org/pub/infozip/) package, you can extract the files using Windows normal line termination conventions with a command similar to:

unzip -aa -a tiff-4.0.5.zip

By default the nmake-based libtiff build does not depend on any additional libraries. Normally libtiff should be built with at least JPEG and ZIP support so that it can open JPEG and ZIP-compressed TIFF files. In order to add additional libraries (e.g. libjpeg, zlib, jbigkit), build those libraries according to their own particular build instructions, and then edit 'nmake.opt' (using a capable plain-text editor) to enable use of the libraries, including specifying where the libraries are installed. It is also necessary to edit libtiff/tiffconf.vc.h to enable the related configuration defines (*JPEG\_SUPPORT*, *OJPEG\_SUPPORT*, *PIXARLOG\_SUPPORT*, *ZIP\_SUPPORT*), or to disable features which are normally included by default. Ignore the comment at the top of the libtiff/tiffconf.vc.h file which says that it has no influence on the build, because the statement is not true for Windows. Please note that the nmake build copies tiffconf.vc.h to tiffconf.h, and copies tif\_config.vc.h to tif\_config.h, overwriting any files which may be present. Likewise, the 'nmake clean' step removes those files.

To build using the provided makefile.vc you may use:

C:\tiff-4.0.5> nmake /f makefile.vc clean  
 C:\tiff-4.0.5> nmake /f makefile.vc  
  
 or (the hard way)  
  
 C:\tiff-4.0.5> cd port  
 C:\tiff-4.0.5\port> nmake /f makefile.vc clean  
 C:\tiff-4.0.5\port> nmake /f makefile.vc  
 C:\tiff-4.0.5> cd ../libtiff  
 C:\tiff-4.0.5\libtiff> nmake /f makefile.vc clean  
 C:\tiff-4.0.5\libtiff> nmake /f makefile.vc  
 C:\tiff-4.0.5\libtiff> cd ..\tools  
 C:\tiff-4.0.5\tools> nmake /f makefile.vc clean  
 C:\tiff-4.0.5\tools> nmake /f makefile.vc

This will build the library file libtiff\libtiff\libtiff.lib.

The makefile also builds a DLL (libtiff.dll) with an associated import library (libtiff\_i.lib). Any builds using libtiff will need to include the LIBTIFF\LIBTIFF directory in the include path.

The libtiff\tools\makefile.vc should build .exe's for all the standard TIFF tool programs.

## Building the Software on a VMS System

The VMS port was done by Karsten Spang ([krs@kampsax.dk](mailto:krs@kampsax.dk)), who also "sort of" maintains it. The VMS specific files are not in the main directories. Instead they are placed under [.CONTRIB.VMS...] in the distribution tree. Installation: It is assumed that you have unpacked the tar file into a VMS directory tree, in this text called DISK:[TIFF].

1. Move the VMS specific files to their proper directories.  
   $ SET DEFAULT DISK:[TIFF.CONTRIB.VMS]  
   $ RENAME [.LIBTIFF]\*.\* [-.-.LIBTIFF]  
   $ RENAME [.TOOLS]\*.\* [-.-.TOOLS]
2. Compile the library.  
   $ SET DEFAULT DISK:[TIFF.LIBTIFF]  
   $ @MAKEVMS
3. Compile the tools.  
   $ SET DEFAULT DISK:[TIFF.TOOLS]  
   $ @MAKEVMS
4. Define the programs.  
   $ DEFINE TIFFSHR DISK:[TIFF.LIBTIFF]TIFFSHR  
   $ FAX2PS :==$DISK:[TIFF.TOOLS]FAX2PS  
   $ FAX2TIFF :==$DISK:[TIFF.TOOLS]FAX2TIFF  
   $ GIF2TIFF :==$DISK:[TIFF.TOOLS]GIF2TIFF  
   $ PAL2RGB :==$DISK:[TIFF.TOOLS]PAL2RGB  
   $ PPM2TIFF :==$DISK:[TIFF.TOOLS]PPM2TIFF  
   $ RAS2TIFF :==$DISK:[TIFF.TOOLS]RAS2TIFF  
   $ RGB2YCBCR :==$DISK:[TIFF.TOOLS]RGB2YCBCR  
   $ THUMBNAIL :==$DISK:[TIFF.TOOLS]THUMBNAIL  
   $ TIFF2BW :==$DISK:[TIFF.TOOLS]TIFF2BW  
   $ TIFF2PS :==$DISK:[TIFF.TOOLS]TIFF2PS  
   $ TIFFCMP :==$DISK:[TIFF.TOOLS]TIFFCMP  
   $ TIFFCP :==$DISK:[TIFF.TOOLS]TIFFCP  
   $ TIFFDITHER:==$DISK:[TIFF.TOOLS]TIFFDITHER  
   $ TIFFDUMP :==$DISK:[TIFF.TOOLS]TIFFDUMP  
   $ TIFFINFO :==$DISK:[TIFF.TOOLS]TIFFINFO  
   $ TIFFMEDIAN:==$DISK:[TIFF.TOOLS]TIFFMEDIAN  
   $ TIFFSPLIT :==$DISK:[TIFF.TOOLS]TIFFSPLIT  
   $ YCBCR :==$DISK:[TIFF.TOOLS]YCBCR

You will want to add these lines to your LOGIN.COM file, after changing the name of the directory that you have used on your machine.

This release has been tested on OpenVMS/VAX 5.5-2, using VAX C 3.2. A previous release was tested under OpenVMS/AXP ?.? using DEC C ?.?, it is believed that this release as well works on AXP. The code contains some GNU C specific things. This does \*not\* imply, however, that the VAX/GCC configuration has been tested, \*it has not\*.

The command procedures (MAKEVMS.COM) for building the library and tools, is believed to choose the correct options for the VAX and AXP cases automatically.

On the AXP, IEEE floating point is used by default. If you want VAX floating point, remove the /FLOAT=IEEE\_FLOAT qualifier, and change HAVE\_IEEEFP=1 to HAVE\_IEEEFP=0 in the MAKEVMS.COM files in both the **libtiff** and **tools** directories.

### Compiling your own program on a VMS system:

When compiling a source file in which you "#include <tiffio.h>", use the following command

$ CC/INCLUDE=DISK:[TIFF.LIBTIFF]

This ensures that the header file is found. On the AXP, also add /FLOAT=IEEE\_FLOAT (if used when building the library).

### Linking your own program to the TIFF library on a VMS system:

You can link to the library in two ways: Either using the shareable library, or using the object library. On the VAX these possibilities are:

1. Using the shareable TIFF library.  
   $ LINK MY\_PROGRAM,DISK:[TIFF.LIBTIFF]TIFF/OPTIONS,SYS$INPUT:/OPTIONS  
    SYS$SHARE:VAXCRTL/SHAREABLE
2. Using the TIFF object library.  
   $ LINK MY\_PROGRAM, -  
    DISK:[TIFF.LIBTIFF]TIFF/LIBRARY/INCLUDE=(TIF\_FAX3SM,TIF\_CODEC), -  
    SYS$INPUT:/OPTIONS  
    SYS$SHARE:VAXCRTL/SHAREABLE

On AXP (and possibly also using DEC C on VAX) the corresponding commands are

1. Using the shareable TIFF library.  
   $ LINK MY\_PROGRAM,DISK:[TIFF.LIBTIFF]TIFF/OPTIONS
2. Using the TIFF object library.  
   $ LINK MY\_PROGRAM,DISK:[TIFF.LIBTIFF]TIFF/LIBRARY

Method 1 uses the shortest link time and smallest .EXE files, but it requires that TIFFSHR is defined as above at link time and **at run time**. Using the compilation procedure above, the tools are linked in this way.

Method 2 gives somewhat longer link time and larger .EXE files, but does not require TIFFSHR to be defined. This method is recommended if you want to run your program on another machine, and for some reason don't want to have the library on that machine. If you plan to have more than one program (including the tools) on the machine, it is recommended that you copy the library to the other machine and use method 1.

## Building the Software on Other Systems

This section contains information that might be useful if you are working on a non-UNIX system that is not directly supported. All library-related files described below are located in the **libtiff** directory.

The library requires two files that are generated *on-the-fly*. The file **tif\_fax3sm.c** has the state tables for the Group 3 and Group 4 decoders. This file is generated by the mkg3states program on a UNIX system; for example,

cd libtiff  
cc -o mkg3states mkg3states.c  
rm -f tif\_fax3sm.c  
./mkg3states -c const tif\_fax3sm.c

The -c option can be used to control whether or not the resutling tables are generated with a const declaration. The -s option can be used to specify a C storage class for the table declarations. The -b option can be used to force data values to be explicitly bracketed with ``{}'' (apparently needed for some MS-Windows compilers); otherwise the structures are emitted in as compact a format as possible. Consult the source code for this program if you have questions.

The second file required to build the library, **version.h**, contains the version information returned by the TIFFGetVersion routine. This file is built on most systems using the mkversion program and the contents of the VERSION and tiff.alpha files; for example,

cd libtiff  
cc -o mkversion mkversion.c  
rm -f version.h  
./mkversion -v ../VERSION -a ../dist/tiff.alpha version.h

Otherwise, when building the library on a non-UNIX system be sure to consult the files **tiffcomp.h** and **tiffconf.h**. The former contains system compatibility definitions while the latter is provided so that the software configuration can be controlled on systems that do not support the make facility for building the software.

Systems without a 32-bit compiler may not be able to handle some of the codecs in the library; especially the Group 3 and 4 decoder. If you encounter problems try disabling support for a particular codec; consult the [documentation](http://docs.google.com/internals.html#Config).

Programs in the tools directory are written to assume an ANSI C compilation environment. There may be a few POSIX'isms as well. The code in the **port** directory is provided to emulate routines that may be missing on some systems. On UNIX systems the configure script automatically figures out which routines are not present on a system and enables the use of the equivalent emulation routines from the **port** directory. It may be necessary to manually do this work on a non-UNIX system.

## Checking out the Software

Assuming you have working versions of tiffgt and tiffsv, you can just use them to view any of the sample images available for testing (see the [section on obtaining the test images](http://docs.google.com/images.html)). Otherwise, you can do a cursory check of the library with the tiffcp and tiffcmp programs. For example,

tiffcp -lzw cramps.tif x.tif  
tiffcmp cramps.tif x.tif

(tiffcmp should be silent if the files compare correctly).

## Table of Contents

The following files makup the core library:

libtiff/tiff.h TIFF spec definitions  
libtiff/tiffcomp.h non-UNIX OS-compatibility definitions  
libtiff/tiffconf.h non-UNIX configuration definitions  
libtiff/tiffio.h public TIFF library definitions  
libtiff/tiffiop.h private TIFF library definitions  
libtiff/t4.h CCITT Group 3/4 code tables+definitions  
libtiff/tif\_dir.h private defs for TIFF directory handling  
libtiff/tif\_fax3.h CCITT Group 3/4-related definitions  
libtiff/tif\_predict.h private defs for Predictor tag support  
libtiff/uvcode.h LogL/LogLuv codec-specific definitions  
libtiff/version.h version string (generated by Makefile)  
libtiff/tif\_apple.c Apple-related OS support  
libtiff/tif\_atari.c Atari-related OS support  
libtiff/tif\_aux.c auxilary directory-related functions  
libtiff/tif\_close.c close an open TIFF file  
libtiff/tif\_codec.c configuration table of builtin codecs  
libtiff/tif\_compress.c compression scheme support  
libtiff/tif\_dir.c directory tag interface code  
libtiff/tif\_dirinfo.c directory known tag support code  
libtiff/tif\_dirread.c directory reading code  
libtiff/tif\_dirwrite.c directory writing code  
libtiff/tif\_dumpmode.c "no" compression codec  
libtiff/tif\_error.c library error handler  
libtiff/tif\_fax3.c CCITT Group 3 and 4 codec  
libtiff/tif\_fax3sm.c G3/G4 state tables (generated by mkg3states)  
libtiff/tif\_flush.c i/o and directory state flushing  
libtiff/tif\_getimage.c TIFFRGBAImage support  
libtiff/tif\_jpeg.c JPEG codec (interface to the IJG distribution)  
libtiff/tif\_luv.c SGI LogL/LogLuv codec  
libtiff/tif\_lzw.c LZW codec  
libtiff/tif\_msdos.c MSDOS-related OS support  
libtiff/tif\_next.c NeXT 2-bit scheme codec (decoding only)  
libtiff/tif\_open.c open and simply query code  
libtiff/tif\_packbits.c Packbits codec  
libtiff/tif\_pixarlog.c Pixar codec  
libtiff/tif\_predict.c Predictor tag support  
libtiff/tif\_print.c directory printing support  
libtiff/tif\_read.c image data reading support  
libtiff/tif\_strip.c some strip-related code  
libtiff/tif\_swab.c byte and bit swapping support  
libtiff/tif\_thunder.c Thunderscan codec (decoding only)  
libtiff/tif\_tile.c some tile-related code  
libtiff/tif\_unix.c UNIX-related OS support  
libtiff/tif\_version.c library version support  
libtiff/tif\_vms.c VMS-related OS support  
libtiff/tif\_warning.c library warning handler  
libtiff/tif\_win3.c Windows-3.1-related OS support  
libtiff/tif\_win32.c Win32 (95/98/NT) related OS support  
libtiff/tif\_write.c image data writing support  
libtiff/tif\_zip.c Deflate codec  
  
libtiff/mkg3states.c program to generate G3/G4 decoder state tables  
libtiff/mkspans.c program to generate black-white span tables  
libtiff/mkversion.c program to generate libtiff/version.h.

Last updated: $Date: 2015-08-29 15:30:11 $