# **Modifying The TIFF Library**

This chapter provides information about the internal structure of the library, how to control the configuration when building it, and how to add new support to the library. The following sections are found in this chapter:

* [Library Configuration](#gjdgxs)
* [General Portability Comments](#30j0zll)
* [Types and Portability](#1fob9te)
* [Adding New Tags](http://docs.google.com/addingtags.html)
* [Adding New Builtin Codecs](#3znysh7)
* [Adding New Codec-private Tags](http://docs.google.com/addingtags.html#AddingCODECTags)
* [Other Comments](#2et92p0)

### **Library Configuration**

Information on compiling the library is given [elsewhere in this documentation](http://docs.google.com/build.html). This section describes the low-level mechanisms used to control the optional parts of the library that are configured at build time. Control is based on a collection of C defines that are specified either on the compiler command line or in a configuration file such as port.h (as generated by the configure script for UNIX systems) or **tiffconf.h**.

Configuration defines are split into three areas:

* those that control which compression schemes are configured as part of the builtin codecs,
* those that control support for groups of tags that are considered optional, and
* those that control operating system or machine-specific support.

If the define COMPRESSION\_SUPPORT is **not defined** then a default set of compression schemes is automatically configured:

* CCITT Group 3 and 4 algorithms (compression codes 2, 3, 4, and 32771),
* the Macintosh PackBits algorithm (compression 32773),
* a 4-bit run-length encoding scheme from ThunderScan (compression 32809),
* a 2-bit encoding scheme used by NeXT (compression 32766), and
* two experimental schemes intended for images with high dynamic range (compression 34676 and 34677).

To override the default compression behaviour define COMPRESSION\_SUPPORT and then one or more additional defines to enable configuration of the appropriate codecs (see the table below); e.g.

#define COMPRESSION\_SUPPORT  
#define CCITT\_SUPPORT  
#define PACKBITS\_SUPPORT  
Several other compression schemes are configured separately from the default set because they depend on ancillary software packages that are not distributed with libtiff.

Support for JPEG compression is controlled by JPEG\_SUPPORT. The JPEG codec that comes with libtiff is designed for use with release 5 or later of the Independent JPEG Group's freely available software distribution. This software can be retrieved from the directory [ftp.uu.net:/graphics/jpeg/](ftp://ftp.uu.net/graphics/jpeg).

 *Enabling JPEG support automatically enables support for the TIFF 6.0 colorimetry and YCbCr-related tags.*

Experimental support for the deflate algorithm is controlled by DEFLATE\_SUPPORT. The deflate codec that comes with libtiff is designed for use with version 0.99 or later of the freely available libz library written by Jean-loup Gailly and Mark Adler. The data format used by this library is described in the files [zlib-3.1.doc](ftp://ftp.uu.net/pub/archiving/zip/doc/zlib-3.1.doc), and [deflate-1.1.doc](ftp://ftp.uu.net/pub/archiving/zip/doc/deflate-1.1.doc), available in the directory [ftp.uu.net:/pub/archiving/zip/doc](ftp://ftp.uu.net/pub/archiving/zip/doc). The library can be retried from the directory ftp.uu.net:/pub/archiving/zip/zlib/ (or try quest.jpl.nasa.gov:/beta/zlib/).

 *The deflate algorithm is experimental. Do not expect to exchange files using this compression scheme; it is included only because the similar, and more common, LZW algorithm is claimed to be governed by licensing restrictions.*

By default **tiffconf.h** defines COLORIMETRY\_SUPPORT, YCBCR\_SUPPORT, and CMYK\_SUPPORT.

|  |  |
| --- | --- |
| Define | Description |
| CCITT\_SUPPORT | CCITT Group 3 and 4 algorithms (compression codes 2, 3, 4, and 32771) |
| PACKBITS\_SUPPORT | Macintosh PackBits algorithm (compression 32773) |
| LZW\_SUPPORT | Lempel-Ziv & Welch (LZW) algorithm (compression 5) |
| THUNDER\_SUPPORT | 4-bit run-length encoding scheme from ThunderScan (compression 32809) |
| NEXT\_SUPPORT | 2-bit encoding scheme used by NeXT (compression 32766) |
| OJPEG\_SUPPORT | obsolete JPEG scheme defined in the 6.0 spec (compression 6) |
| JPEG\_SUPPORT | current JPEG scheme defined in TTN2 (compression 7) |
| ZIP\_SUPPORT | experimental Deflate scheme (compression 32946) |
| PIXARLOG\_SUPPORT | Pixar's compression scheme for high-resolution color images (compression 32909) |
| SGILOG\_SUPPORT | SGI's compression scheme for high-resolution color images (compression 34676 and 34677) |
| COLORIMETRY\_SUPPORT | support for the TIFF 6.0 colorimetry tags |
| YCBCR\_SUPPORT | support for the TIFF 6.0 YCbCr-related tags |
| CMYK\_SUPPORT | support for the TIFF 6.0 CMYK-related tags |
| ICC\_SUPPORT | support for the ICC Profile tag; see *The ICC Profile Format Specification*, Annex B.3 "Embedding ICC Profiles in TIFF Files"; available at <http://www.color.org> |

### **General Portability Comments**

This software is developed on Silicon Graphics UNIX systems (big-endian, MIPS CPU, 32-bit ints, IEEE floating point). The configure shell script generates the appropriate include files and make files for UNIX systems. Makefiles exist for non-UNIX platforms that the code runs on -- this work has mostly been done by other people.

In general, the code is guaranteed to work only on SGI machines. In practice it is highly portable to any 32-bit or 64-bit system and much work has been done to insure portability to 16-bit systems. If you encounter portability problems please return fixes so that future distributions can be improved.

The software is written to assume an ANSI C compilation environment. If your compiler does not support ANSI function prototypes, const, and <stdarg.h> then you will have to make modifications to the software. In the past I have tried to support compilers without const and systems without <stdarg.h>, but I am *no longer interested in these antiquated environments*. With the general availability of the freely available GCC compiler, I see no reason to incorporate modifications to the software for these purposes.

An effort has been made to isolate as many of the operating system-dependencies as possible in two files: **tiffcomp.h** and **libtiff/tif\_<os>.c**. The latter file contains operating system-specific routines to do I/O and I/O-related operations. The UNIX (**tif\_unix.c**), Macintosh (**tif\_apple.c**), and VMS (**tif\_vms.c**) code has had the most use; the MS/DOS support (**tif\_msdos.c**) assumes some level of UNIX system call emulation (i.e. open, read, write, fstat, malloc, free).

Native CPU byte order is determined on the fly by the library and does not need to be specified. The HOST\_FILLORDER and HOST\_BIGENDIAN definitions are not currently used, but may be employed by codecs for optimization purposes.

The following defines control general portability:

|  |  |
| --- | --- |
| BSDTYPES | Define this if your system does NOT define the usual BSD typedefs: u\_char, u\_short, u\_int, u\_long. |
| HAVE\_IEEEFP | Define this as 0 or 1 according to the floating point format suported by the machine. If your machine does not support IEEE floating point then you will need to add support to tif\_machdep.c to convert between the native format and IEEE format. |
| HAVE\_MMAP | Define this if there is *mmap-style* support for mapping files into memory (used only to read data). |
| HOST\_FILLORDER | Define the native CPU bit order: one of FILLORDER\_MSB2LSB or FILLORDER\_LSB2MSB |
| HOST\_BIGENDIAN | Define the native CPU byte order: 1 if big-endian (Motorola) or 0 if little-endian (Intel); this may be used in codecs to optimize code |

On UNIX systems HAVE\_MMAP is defined through the running of the configure script; otherwise support for memory-mapped files is disabled. Note that **tiffcomp.h** defines HAVE\_IEEEFP to be 1 (BSDTYPES is not defined).

### **Types and Portability**

The software makes extensive use of C typedefs to promote portability. Two sets of typedefs are used, one for communication with clients of the library and one for internal data structures and parsing of the TIFF format. There are interactions between these two to be careful of, but for the most part you should be able to deal with portability purely by fiddling with the following machine-dependent typedefs:

|  |  |  |
| --- | --- | --- |
| uint8 | 8-bit unsigned integer | tiff.h |
| int8 | 8-bit signed integer | tiff.h |
| uint16 | 16-bit unsigned integer | tiff.h |
| int16 | 16-bit signed integer | tiff.h |
| uint32 | 32-bit unsigned integer | tiff.h |
| int32 | 32-bit signed integer | tiff.h |
| dblparam\_t | promoted type for floats | tiffcomp.h |

(to clarify dblparam\_t, it is the type that float parameters are promoted to when passed by value in a function call.)

The following typedefs are used throughout the library and interfaces to refer to certain objects whose size is dependent on the TIFF image structure:

|  |  |
| --- | --- |
| typedef unsigned int ttag\_t; | directory tag |
| typedef uint16 tdir\_t; | directory index |
| typedef uint16 tsample\_t; | sample number |
| typedef uint32 tstrip\_t; | strip number |
| typedef uint32 ttile\_t; | tile number |
| typedef int32 tsize\_t; | i/o size in bytes |
| typedef void\* tdata\_t; | image data ref |
| typedef void\* thandle\_t; | client data handle |
| typedef int32 toff\_t; | file offset (should be off\_t) |
| typedef unsigned char\* tidata\_t; | internal image data |

Note that tstrip\_t, ttile\_t, and tsize\_t are constrained to be no more than 32-bit quantities by 32-bit fields they are stored in in the TIFF image. Likewise tsample\_t is limited by the 16-bit field used to store the SamplesPerPixel tag. tdir\_t constrains the maximum number of IFDs that may appear in an image and may be an arbitrary size (without penalty). ttag\_t must be either int, unsigned int, pointer, or double because the library uses a varargs interface and ANSI C restricts the type of the parameter before an ellipsis to be a promoted type. toff\_t is defined as int32 because TIFF file offsets are (unsigned) 32-bit quantities. A signed value is used because some interfaces return -1 on error (sigh). Finally, note that tidata\_t is used internally to the library to manipulate internal data. User-specified data references are passed as opaque handles and only cast at the lowest layers where their type is presumed.

### **General Comments**

The library is designed to hide as much of the details of TIFF from applications as possible. In particular, TIFF directories are read in their entirety into an internal format. Only the tags known by the library are available to a user and certain tag data may be maintained that a user does not care about (e.g. transfer function tables).

### **Adding New Builtin Codecs**

To add builtin support for a new compression algorithm, you can either use the "tag-extension" trick to override the handling of the TIFF Compression tag (see [Adding New Tags](http://docs.google.com/addingtags.html)), or do the following to add support directly to the core library:

1. Define the tag value in **tiff.h**.
2. Edit the file **tif\_codec.c** to add an entry to the \_TIFFBuiltinCODECS array (see how other algorithms are handled).
3. Add the appropriate function prototype declaration to **tiffiop.h** (close to the bottom).
4. Create a file with the compression scheme code, by convention files are named **tif\_\*.c** (except perhaps on some systems where the tif\_ prefix pushes some filenames over 14 chars.
5. Edit **Makefile.in** (and any other Makefiles) to include the new source file.

A codec, say foo, can have many different entry points:

TIFFInitfoo(tif, scheme)/\* initialize scheme and setup entry points in tif \*/  
fooSetupDecode(tif) /\* called once per IFD after tags has been frozen \*/  
fooPreDecode(tif, sample)/\* called once per strip/tile, after data is read,  
 but before the first row is decoded \*/  
fooDecode\*(tif, bp, cc, sample)/\* decode cc bytes of data into the buffer \*/  
 fooDecodeRow(...) /\* called to decode a single scanline \*/  
 fooDecodeStrip(...) /\* called to decode an entire strip \*/  
 fooDecodeTile(...) /\* called to decode an entire tile \*/  
fooSetupEncode(tif) /\* called once per IFD after tags has been frozen \*/  
fooPreEncode(tif, sample)/\* called once per strip/tile, before the first row in  
 a strip/tile is encoded \*/  
fooEncode\*(tif, bp, cc, sample)/\* encode cc bytes of user data (bp) \*/  
 fooEncodeRow(...) /\* called to decode a single scanline \*/  
 fooEncodeStrip(...) /\* called to decode an entire strip \*/  
 fooEncodeTile(...) /\* called to decode an entire tile \*/  
fooPostEncode(tif) /\* called once per strip/tile, just before data is written \*/  
fooSeek(tif, row) /\* seek forwards row scanlines from the beginning  
 of a strip (row will always be >0 and <rows/strip \*/  
fooCleanup(tif) /\* called when compression scheme is replaced by user \*/

Note that the encoding and decoding variants are only needed when a compression algorithm is dependent on the structure of the data. For example, Group 3 2D encoding and decoding maintains a reference scanline. The sample parameter identifies which sample is to be encoded or decoded if the image is organized with PlanarConfig=2 (separate planes). This is important for algorithms such as JPEG. If PlanarConfig=1 (interleaved), then sample will always be 0.

### **Other Comments**

The library handles most I/O buffering. There are two data buffers when decoding data: a raw data buffer that holds all the data in a strip, and a user-supplied scanline buffer that compression schemes place decoded data into. When encoding data the data in the user-supplied scanline buffer is encoded into the raw data buffer (from where it is written). Decoding routines should never have to explicitly read data -- a full strip/tile's worth of raw data is read and scanlines never cross strip boundaries. Encoding routines must be cognizant of the raw data buffer size and call TIFFFlushData1() when necessary. Note that any pending data is automatically flushed when a new strip/tile is started, so there's no need do that in the tif\_postencode routine (if one exists). Bit order is automatically handled by the library when a raw strip or tile is filled. If the decoded samples are interpreted by the decoding routine before they are passed back to the user, then the decoding logic must handle byte-swapping by overriding the tif\_postdecode routine (set it to TIFFNoPostDecode) and doing the required work internally. For an example of doing this look at the horizontal differencing code in the routines in **tif\_predict.c**.

The variables tif\_rawcc, tif\_rawdata, and tif\_rawcp in a TIFF structure are associated with the raw data buffer. tif\_rawcc must be non-zero for the library to automatically flush data. The variable tif\_scanlinesize is the size a user's scanline buffer should be. The variable tif\_tilesize is the size of a tile for tiled images. This should not normally be used by compression routines, except where it relates to the compression algorithm. That is, the cc parameter to the tif\_decode\* and tif\_encode\* routines should be used in terminating decompression/compression. This ensures these routines can be used, for example, to decode/encode entire strips of data.

In general, if you have a new compression algorithm to add, work from the code for an existing routine. In particular, **tif\_dumpmode.c** has the trivial code for the "nil" compression scheme, **tif\_packbits.c** is a simple byte-oriented scheme that has to watch out for buffer boundaries, and **tif\_lzw.c** has the LZW scheme that has the most complexity -- it tracks the buffer boundary at a bit level. Of course, using a private compression scheme (or private tags) limits the portability of your TIFF files.

Last updated: $Date: 2016-09-25 20:05:44 $