# **Defining New TIFF Tags**

Libtiff has built-in knowledge of all the standard TIFF tags, as well as extentions. The following describes how to add knowledge of new tags as builtins to libtiff, or how to application specific tags can be used by applications without modifying libtiff.

## **TIFFFieldInfo**

How libtiff manages specific tags is primarily controlled by the definition for that tag value stored internally as a TIFFFieldInfo structure. This structure looks like this:

typedef struct {  
 ttag\_t field\_tag; /\* field's tag \*/  
 short field\_readcount; /\* read count/TIFF\_VARIABLE/TIFF\_SPP \*/  
 short field\_writecount; /\* write count/TIFF\_VARIABLE \*/  
 TIFFDataType field\_type; /\* type of associated data \*/  
 unsigned short field\_bit; /\* bit in fieldsset bit vector \*/  
 unsigned char field\_oktochange;/\* if true, can change while writing \*/  
 unsigned char field\_passcount;/\* if true, pass dir count on set \*/  
 char \*field\_name; /\* ASCII name \*/  
} TIFFFieldInfo;

* **field\_tag**: the tag number. For instance 277 for the SamplesPerPixel tag. Builtin tags will generally have a #define in tiff.h for each known tag.
* **field\_readcount**: The number of values which should be read. The special value TIFF\_VARIABLE (-1) indicates that a variable number of values may be read. The special value TIFFTAG\_SPP (-2) indicates that there should be one value for each sample as defined by TIFFTAG\_SAMPLESPERPIXEL. The special value TIFF\_VARIABLE2 (-3) is presumably similar to TIFF\_VARIABLE though I am not sure what the distinction in behaviour is. This field is TIFF\_VARIABLE for variable length ascii fields.
* **field\_writecount**: The number of values which should be written. Generally the same as field\_readcount. A few built-in exceptions exist, but I haven't analysed why they differ.
* **field\_type**: Type of the field. One of TIFF\_BYTE, TIFF\_ASCII, TIFF\_SHORT, TIFF\_LONG, TIFF\_RATIONAL, TIFF\_SBYTE, TIFF\_UNDEFINED, TIFF\_SSHORT, TIFF\_SLONG, TIFF\_SRATIONAL, TIFF\_FLOAT, TIFF\_DOUBLE or TIFF\_IFD. Note that some fields can support more than one type (for instance short and long). These fields should have multiple TIFFFieldInfos.
* **field\_bit**: Built-in tags stored in special fields in the TIFF structure have assigned field numbers to distinguish them (ie. FIELD\_SAMPLESPERPIXEL). New tags should generally just use FIELD\_CUSTOM indicating they are stored in the generic tag list.
* **field\_oktochange**: TRUE if it is OK to change this tag value while an image is being written. FALSE for stuff that must be set once and then left unchanged (like ImageWidth, or PhotometricInterpretation for instance).
* **field\_passcount**: If TRUE, then the count value must be passed in TIFFSetField(), and TIFFGetField(), otherwise the count is not required. This should generally be TRUE for non-ascii variable count tags unless the count is implicit (such as with the colormap).
* **field\_name**: A name for the tag. Normally mixed case (studly caps) like "StripByteCounts" and relatively short.

A TIFFFieldInfo definition exists for each built-in tag in the tif\_dirinfo.c file. Some tags which support multiple data types have more than one definition, one per data type supported.

Various functions exist for getting the internal TIFFFieldInfo definitions, including \_TIFFFindFieldInfo(), and \_TIFFFindFieldInfoByName(). See tif\_dirinfo.c for details. There must be some mechanism to get the whole list, though I don't see it off hand.

## **Default Tag Auto-registration**

In libtiff 3.6.0 a new mechanism was introduced allowing libtiff to read unrecognised tags automatically. When an unknown tags is encountered, it is automatically internally defined with a default name and a type derived from the tag value in the file. Applications only need to predefine application specific tags if they need to be able to set them in a file, or if particular calling conventions are desired for TIFFSetField() and TIFFGetField().

When tags are autodefined like this the **field\_readcount** and **field\_writecount** values are always TIFF\_VARIABLE. The **field\_passcount** is always TRUE, and the **field\_bit** is FIELD\_CUSTOM. The field name will be "Tag %d" where the %d is the tag number.

## **Defining Application Tags**

For various reasons, it is common for applications to want to define their own tags to store information outside the core TIFF specification. This is done by calling TIFFMergeFieldInfo() with one or more TIFFFieldInfos.

The libgeotiff library provides geospatial information extentions within a TIFF file. First, a set of TIFFFieldInfo's is prepared with information on the new tags:

static const TIFFFieldInfo xtiffFieldInfo[] = {  
   
 /\* XXX Insert Your tags here \*/  
 { TIFFTAG\_GEOPIXELSCALE, -1,-1, TIFF\_DOUBLE, FIELD\_CUSTOM,  
 TRUE, TRUE, "GeoPixelScale" },  
 { TIFFTAG\_GEOTRANSMATRIX, -1,-1, TIFF\_DOUBLE, FIELD\_CUSTOM,  
 TRUE, TRUE, "GeoTransformationMatrix" },  
 { TIFFTAG\_GEOTIEPOINTS, -1,-1, TIFF\_DOUBLE, FIELD\_CUSTOM,  
 TRUE, TRUE, "GeoTiePoints" },  
 { TIFFTAG\_GEOKEYDIRECTORY, -1,-1, TIFF\_SHORT, FIELD\_CUSTOM,  
 TRUE, TRUE, "GeoKeyDirectory" },  
 { TIFFTAG\_GEODOUBLEPARAMS, -1,-1, TIFF\_DOUBLE, FIELD\_CUSTOM,  
 TRUE, TRUE, "GeoDoubleParams" },  
 { TIFFTAG\_GEOASCIIPARAMS, -1,-1, TIFF\_ASCII, FIELD\_CUSTOM,  
 TRUE, FALSE, "GeoASCIIParams" }  
};

In order to define the tags, we call TIFFMergeFieldInfo() on the desired TIFF handle with the list of TIFFFieldInfos.

#define N(a) (sizeof (a) / sizeof (a[0]))  
  
 /\* Install the extended Tag field info \*/  
 TIFFMergeFieldInfo(tif, xtiffFieldInfo, N(xtiffFieldInfo));

The tags need to be defined for each TIFF file opened - and when reading they should be defined before the tags of the file are read, yet a valid TIFF \* is needed to merge the tags against. In order to get them registered at the appropriate part of the setup process, it is necessary to register our merge function as an extender callback with libtiff. This is done with TIFFSetTagExtender(). We also keep track of the previous tag extender (if any) so that we can call it from our extender allowing a chain of customizations to take effect.

static TIFFExtendProc \_ParentExtender = NULL;  
  
static  
void \_XTIFFInitialize(void)  
{  
 static int first\_time=1;  
   
 if (! first\_time) return; /\* Been there. Done that. \*/  
 first\_time = 0;  
   
 /\* Grab the inherited method and install \*/  
 \_ParentExtender = TIFFSetTagExtender(\_XTIFFDefaultDirectory);  
}

The extender callback is looks like this. It merges in our new fields and then calls the next extender if there is one in effect.

static void  
\_XTIFFDefaultDirectory(TIFF \*tif)  
{  
 /\* Install the extended Tag field info \*/  
 TIFFMergeFieldInfo(tif, xtiffFieldInfo, N(xtiffFieldInfo));  
  
 /\* Since an XTIFF client module may have overridden  
 \* the default directory method, we call it now to  
 \* allow it to set up the rest of its own methods.  
 \*/  
  
 if (\_ParentExtender)   
 (\*\_ParentExtender)(tif);  
}

The above approach ensures that our new definitions are used when reading or writing any TIFF file. However, since on reading we already have default definitions for tags, it is usually not critical to pre-define them. If tag definitions are only required for writing custom tags, you can just call TIFFMergeFieldInfo() before setting new tags. The whole extender architecture can then be avoided.

## **Adding New Builtin Tags**

A similar approach is taken to the above. However, the TIFFFieldInfo should be added to the tiffFieldInfo[] list in tif\_dirinfo.c. Ensure that new tags are added in sorted order by the tag number.

Normally new built-in tags should be defined with FIELD\_CUSTOM; however, if it is desirable for the tag value to have it's own field in the TIFFDirectory structure, then you will need to #define a new FIELD\_ value for it, and add appropriate handling as follows:

1. Define the tag in **tiff.h**.
2. Add a field to the directory structure in **tif\_dir.h** and define a FIELD\_\* bit (also update the definition of FIELD\_CODEC to reflect your addition).
3. Add an entry in the TIFFFieldInfo array defined at the top of **tif\_dirinfo.c**. Note that you must keep this array sorted by tag number and that the widest variant entry for a tag should come first (e.g. LONG before SHORT).
4. Add entries in \_TIFFVSetField() and \_TIFFVGetField() for the new tag.
5. (*optional*) If the value associated with the tag is not a scalar value (e.g. the array for TransferFunction) and requires special processing, then add the appropriate code to TIFFReadDirectory() and TIFFWriteDirectory(). You're best off finding a similar tag and cribbing code.
6. Add support to TIFFPrintDirectory() in **tif\_print.c** to print the tag's value.

If you want to maintain portability, beware of making assumptions about data types. Use the typedefs (uint16, etc. when dealing with data on disk and t\*\_t when stuff is in memory) and be careful about passing items through printf or similar vararg interfaces.

## **Adding New Codec-private Tags**

To add tags that are meaningful *only when a particular compression algorithm is used* follow these steps:

1. Define the tag in **tiff.h**.
2. Allocate storage for the tag values in the private state block of the codec.
3. Insure the state block is created when the codec is initialized.
4. At TIFFInitfoo time override the method pointers in the TIFF structure for getting, setting and printing tag values. For example,  
    sp->vgetparent = tif->tif\_vgetfield;  
    tif->tif\_vgetfield = fooVGetField; /\* hook for codec tags \*/  
    sp->vsetparent = tif->tif\_vsetfield;  
    tif->tif\_vsetfield = fooVSetField; /\* hook for codec tags \*/  
    tif->tif\_printdir = fooPrintDir; /\* hook for codec tags \*/  
   (Actually you may decide not to override the tif\_printdir method, but rather just specify it).
5. Create a private TIFFFieldInfo array for your tags and merge them into the core tags at initialization time using \_TIFFMergeFieldInfo; e.g.  
    \_TIFFMergeFieldInfo(tif, fooFieldInfo, N(fooFieldInfo));  
   (where N is a macro used liberaly throughout the distributed code).
6. Fill in the get and set routines. Be sure to call the parent method for tags that you are not handled directly. Also be sure to set the FIELD\_\* bits for tags that are to be written to the file. Note that you can create ``pseudo-tags'' by defining tags that are processed exclusively in the get/set routines and never written to file (see the handling of TIFFTAG\_FAXMODE in **tif\_fax3.c** for an example of this).
7. Fill in the print routine, if appropriate.

Note that space has been allocated in the FIELD\_\* bit space for codec-private tags. Define your bits as FIELD\_CODEC+<offset> to keep them away from the core tags. If you need more tags than there is room for, just increase FIELD\_SETLONGS at the top of **tiffiop.h**.Last updated: $Date: 2016-09-25 20:05:44 $