**PyrTIFF: a TIFF specification extension for storing multi-resolution pyramids**

TIFF (and its derivatives) is one of the most heavily used image file formats for scientific image data. Other formats, arguably more suited for scientific image data such as HDF5, have not yet achieved adoption rates anywhere near that of TIFF.

Unlike other popular image formats (JPEG, PNG, GIF, etc.), the flexibility of TIFF enables storage of complex image data types such as high bit-depth data, multi-channel data, Z-stacks, time series, etc. and furthermore some TIFF derivatives (such as OME-TIFF) have codified a standardized way to store such data beyond the official TIFF specification.

The TIFF specification (v6.0 from 1992: <https://www.awaresystems.be/imaging/tiff/specification/TIFF6.pdf>) is officially owned by Adobe Systems Inc., but they have not touched the specification since 2002 (<https://www.awaresystems.be/imaging/tiff/specification/TIFFphotoshop.pdf>). An open source-minded group of volunteers have effectively managed a *de facto* evolved TIFF specification (e.g. incorporating the BigTIFF extension). This activity is loosely organized around the libtiff library (<http://www.simplesystems.org/libtiff/> ). Thus we can pretend that TIFF is an open standard with libtiff as the open source reference implementation.

One critical needed capability, namely a way to store multi-resolution pyramids is needed for large XY-extent images such as digital pathology images, and that has not been standardized. A number of *ad hoc* implementations exist in open source (ImageMagick and VIPS), as well as commercially (Aperio’s SVS, Hamamatsu’s NDPI, Leica’s SCN, Ventana’s BIF/TIF, etc. However, none of these implementations are compatible with each other and most are not well-specified or documented and the commercial ones tend to be proprietary. Furthermore, all of these *ad hoc* approaches all impose significant and different restrictions on TIFF’s flexibility. **We need an open but standardized specification and implementations thereof that support the flexibility needed for scientific image data.**

**The following is an attempt to define an extension to the TIFF specification that combines most of the desirable features of a number of existing extensions and derivations while keeping maximum compatibility with the existing TIFF 6.0 specification and with existing implementations and derivations. We’ll refer to this specification extension as “pyrTIFF”.**

1. While both TIFF and BigTIFF encodings are allowed, large XY-extent image data will likely require use of the BigTIFF extension and the method defined in libtiff 4.x is required for that. See: <https://www.awaresystems.be/imaging/tiff/bigtiff.html>
2. For any data that includes multi-channel, Z stack, time series, or other dimensionality extension, the OME-TIFF specification is highly recommended. See: <https://docs.openmicroscopy.org/ome-model/5.5.7/ome-tiff/specification.html>
3. For any image data that requires rich metadata, the OME-TIFF method and schema should be strongly considered if at all appropriate for the image data type. Alternative metadata options include EXIF for photography data or GeoTIFF metadata definitions for geo-referencing data.
4. In the TIFF specification, compression is done on a per-tile basis. A variety of compression methods are optionally allowed incl. JPEG, LZW, deflate (i.e. PNG or zlib), JPEG2000, JPEG-XR, etc. Also allowed is compression/encoding on a per plane basis (i.e. per IFD) specifically for inherently multi-resolution compression/encoding methods such as JPEG2000. (**To be revised:** Note: if compression is per tile, it is allowed, and arguably desirable for scientific image data, that the full-resolution image is stored using lossless or no compression while the sub-resolution tiles may be stored using lossy compression for improved reading and rendering speed.)
5. A specific question is where to store the pyramid of sub-resolution representations per image plane. The 2 possibilities are: option 1) follow the implied intent of the use of SubIFDs (<https://www.awaresystems.be/imaging/tiff/specification/TIFFPM6.pdf>) and store the sub-resolution representations in a sequence of SubIFDs of the image plane’s top-level IFD, and option 2) stay in line with almost every implementation of pyramid storage so far and store the reduced resolution images as a sequence of top-level IFDs. Option 1 more closely follows Adobe’s original intent and is more elegant and while it reduces compatibility with existing approaches, we propose to initially support only option 1. A future revision may also accept option 2 and reader code may accept either option.
6. In either case the reduced resolution sequence preferentially follows a dyadic reduction in both X and Y until one of the dimensions reaches 256. Other reduction schemes are allowed and should be encoded in a higher level specification such as the OME-XML header or alternatively the reader code can deduce it from the sizes of the reduced resolution images in the sequence.
7. As already formalized in the TIFF spec, each full-resolution IFD should have bit 0 of the NewSubfileType tag unset while each reduced resolution (Sub)IFD should have bit 0 of the NewSubfileType tag set. The deprecated SubfileType tag should not be used by the writer and should be ignored by a reader.
8. For general-purpose use, a thumbnail image can optionally be stored in an EXIF IFD as defined in <http://www.exif.org/Exif2-2.PDF>. Additional optional utility images such as a more flexible thumbnail image, a slide label image, and a macro image to support e.g. digital pathology applications can be stored in top-level IFDs and annotated with higher-level metadata as explained in 2) and 3) above. Such images must be JPEG or PNG (or APNG/GIF??), strip or raster (no tiles), and no larger than 4096x4096; recommended size is 1024 pixels on largest side.

**Implementations:**

**Currently existing TIFF readers/writers:** When presented with pyrTIFF-compliant TIFF file, most existing TIFF readers will at a minimum be able to read the initial full resolution image from the file and ignore the sub-resolution versions or alternatively treat those as additional separate images in the file. This is considered acceptable behavior. A few readers such as the IIPImage server (<http://iipimage.sourceforge.net/>) and the OpenSlide library (<http://openslide.org/>) might with some restrictions be able to use the full pyramid of sub-resolutions. A number of widely used software applications (ImageMagick, GraphicsMagick, VIPS, some versions of Adobe Photoshop) can generate pyramidal TIFF files that are not compatible with the specifications proposed but instead use the option 2 scheme in point 5) above. Most TIFF-derived digital pathology formats (Aperio’s SVS, Leica’s SCN, Ventana’s BIF, etc.) also use some version of the option 2 scheme; thus their readers/writers are not compatible.

**OME-TIFF:** Currently the OME-TIFF specification does not cover storing pyramidal large images. With this write-up we propose that the OME-TIFF specification be extended to support such images. In doing so, the pyrTIFF extension would become an integral part of the OME-TIFF specification. One approach could be as follows: Use the OME-XML TiffData element to define/annotate the reduced resolution SubIFDs (e.g. <TiffData IFD=”1” Encoding=”Pyramid”/> ). For non-dyadic reduction schemes, the TiffData element can explicitly specify the sub-resolution scheme (e.g.: <TiffData IFD=”1” Encoding=”Pyramid” Subresolutions=”1.0, 0.25, 0.125”/> ) or assume the scheme used can be deduced from the X-Y sizes of the SubIFD images. For encoding/compression schemes that incorporate pyramids natively such as JPEG2000, the TiffData element can be used to flag that (e.g.: <TiffData IFD=”1” Encoding=”JP2000”/> ). Optional utility images such as thumbnails, slide labels, or macro images can be specified similarly (e.g. <TiffData IFD=”5” Encoding=”Thumbnail”/> ).

**Bio-Formats, OMERO, and OME-Files:** These products/projects from the OME already have some support for pyramidal large image types as an input and/or for internal storage. As part of this write-up, we propose the following developments:
- Bio-Formats: a) extend (or subclass) the existing tiff reader to accept and handle pyrTIFF files and treat them similarly to how it handles Aperio SVS files. b) extend (or subclass) the OME-TIFF writer (and possibly also the regular tiff writer) to optionally generate pyrTIFF-compliant OME-TIFF files (see above) where the pyramid can be calculated at time of writing, or used from the internal OMERO Pyramid format, or, possibly in the case of bfconvert, used from an already pyramid’ed input file.
- OMERO: accept pyrTIFF-compliant files as just another pyramidal file format providing such image files with fast import, no pyramid calculation, and no additional storage needs beyond the original file. It appears from the documentation that the internal OMERO Pyramid format v1.0.0 may be compatible with pyrTIFF and could thus be stored in the proposed extended OME-TIFF format for external use.
- OME Files: this C++ reference library for OME-TIFF file format needs to be extended to implement the OME-TIFF w/ pyrTIFF extension specification for both reading and writing. Ideally the regular tiff reading and writing would also support pyrTIFF-compliant files.

**Libtiff:** this general-purpose reference C/C++ library for reading and writing TIFF files has, as mentioned above, become the vehicle for the *de facto* extended specification of the TIFF format. We propose that the libtiff community consider accepting the pyrTIFF extension in a similar way to how it incorporated BigTIFF. In practice this would mean: a) the libtiff documentation to incorporate the pyrTIFF extension, b) development of a few wrapper functions as part of libtiff that implement: high-level read functions to access the desired subresolution level or utility images, high-level write functions which optionally generate the sub-resolution pyramid, and high-level write functions for the utility images. Even without b) the current libtiff is already fully capable of writing pyrTIFF-compliant TIFF files and it can read all such files that don’t rely on features not (yet) supported by libtiff.