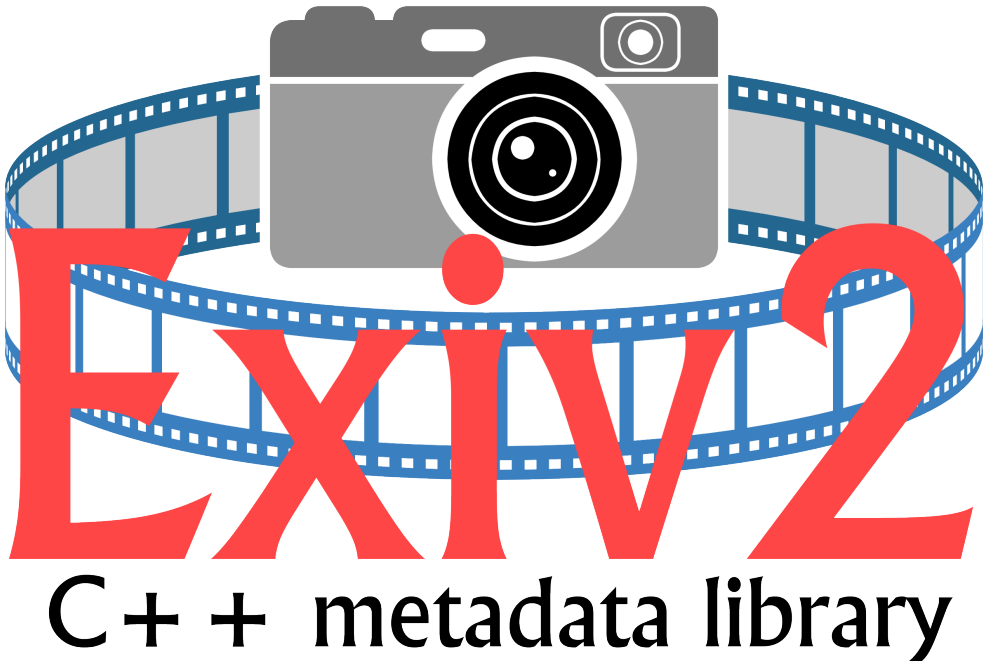


Image Metadata

and

Exiv2 Architecture

Robin Mills

2020-12-05

Dedication and Acknowledgment

*I want to say **Thank You** to a few folks who have made this book possbile.*

First, my wife Alison, who has been my loyal support since the day we met in High School in 1967.

Secondly, Andreas Huggel the founder of the project and Luis and Dan who have worked tirelessly with me since 2017.

Exiv2 contributors (in alphabetical order): Abhinav, Alan, Andreas (both of them), Arnold, Ben, Gilles, Kevin, Leo, Leonardo, Mahesh, Michał, Mikayel, Miloš, Nehal, Neils, Phil, Rosen, Sridhar, Thomas, Tuan …. and others who have contributed to Exiv2.

File Detectives: Phil Harvey, Dave Coffin, Laurent Clévy.

And our cat Lizzie.

TABLE of CONTENTS

About this book

This book is about **Image Metadata *and* Exiv2 Architecture**.

Image Metadata is the information stored in a digital image in addition to the image itself. Data such as the camera model, date, time, location and camera settings are stored. To my knowledge, no book has been written about this important technology.

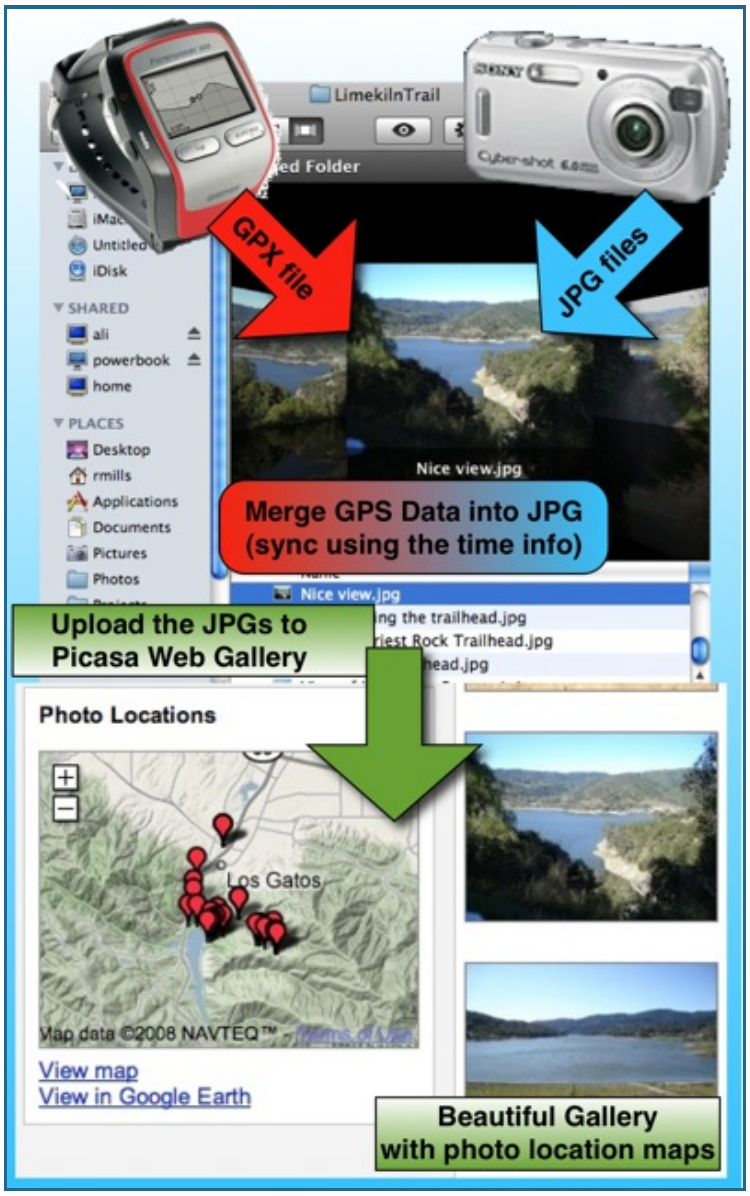
Exiv2 Architecture is about the Exiv2 library and command-line application which implements cross-platform code in C++ to read, modify, insert and delete items of metadata. I've been working on this code since 2008 and, as I approach my 70th birthday, would like to document my knowledge in the hope that the code will be maintained and developed by others in future.

At the moment, the book is ***work in progress*** and expected to be finished by the end of 2020. Exiv2 v0.27.3 shipped on schedule on 2020-06-30 and I feel the text and the code discussed in this book are good enough to be released in its current state.

TOC

How did I get interested in this matter?

I first became interested in metadata because of a trail conversation with Dennis Connor in 2008. Dennis and I ran frequently together in Silicon Valley and Dennis was a Software Development Manager in a company that made GPS systems for Precision Agriculture. I had a Garmin Forerunner 201 Watch. We realised that we could extract the GPS data from the watch in GPX format, then merge the position into photos. Today this is called "GeoTagging" and is supported by many applications.



I said "Oh, it can't be too difficult to do that!". And here we are more than a decade later still working on the project. The program geotag.py was completed in about 6 weeks. Most of the effort went into porting Exiv2 and pyexiv2 to Visual Studio and macOS. Both Exiv2 and pyexiv2 were Linux only at that time.

The program samples/geotag.cpp is a command line utility to geotag photos and I frequently use this on my own photographs. Today, I have a Samsung Galaxy Watch which uploads runs to Strava. I download the GPX from Strava. The date/time information in the JPG is the key to search for the position data. The GPS tags are created and saved in the image.

In 2008, I chose to implement this in python because I wanted to learn the language. Having discovered exiv2 and the python wrapper pyexiv2, I set off with enthusiasm to build a cross-platform script to run on **Windows** *(XP, Visual Studio 2003)*, **Ubuntu Linux** *(Hardy Heron 2008.04 LTS)* and **macOS** *(32 bit Tiger 10.4 on a big-endian PPC)*. After I finished, I emailed Andreas. He responded in less than an hour and invited me to join Team Exiv2. Initialially, I provided support to build Exiv2 with Visual Studio.

Incidentally, later in 2008, Dennis offered me a contract to port his company's Linux code to Visual Studio to be

used on a Windows CE Embedded Controller. 1 million lines of C++ were ported from Linux in 6 weeks. I worked with Dennis for 4 years on all manner of GPS related software development.

https://clanmills.com/articles/gpsexiftags/

I have never been employed to work on Metadata. I was a Senior Computer Scientist at Adobe for more than 10 years, however I was never involved with XMP or Metadata.

TOC

2012 - 2017

By 2012, Andreas was losing interest in Exiv2. Like all folks, he has many matters which deserve his time. A family, a business, biking and other pursuits. From 2012 until 2017, I supported Exiv2 mostly alone. I had lots of encouragement from Alan and other occasional contributors. Neils did great work on lens recognition and compatibility with ExifTool. Ben helped greatly with WebP support and managed the transition of the code from SVN to GitHub. Phil *(of ExifTool fame)* has always been very supportive and helpful.

I must also mention our adventures with Google Summer of Code and our students Abhinav, Tuan and Mahesh. GSoC is a program at Google to sponsor students to contribute to open source projects. 1200 Students from around the world are given a bounty of $5000 to contribute 500 hours to a project during summer recess. The projects are supervised by a mentor. Exiv2 is considered to be part of the KDE family of projects. Within KDE, there is a sub-group of Graphics Applications and Technology. We advertised our projects, the students wrote proposals and some were accepted by Google on the Recommendation of the KDE/Graphics group.

In 2012, Abhinav joined us and contributed the Video read code and was mentored by Andreas. In 2013, Tuan joined us and contributed the WebReady code and was mentored by me. Mahesh also joined us to contribute the Video write code and was mentored by Abhinav.

I personally found working with the students to be enjoyable and interesting. I retired from work in 2014 and returned to England after 15 years in Silicon Valley. In 2016, Alison and I had a trip round the world and spent a day with Mahesh in Bangalore and with Tuan in Singapore. We were invited to stay with Andreas and his family. We subsequently went to Vietnam to attend Tuan's wedding in 2017.

TOC

2017 - Present (2021)

After v0.26 was released in 2017, Luis and Dan started making contributions. They have made many important contributions in the areas of security, test and build. In 2019, Kevin joined us. He discovered and fixed some security issues.

The current release of Exiv2 is v0.27.3 and shipped on 2020-06-30. I hope v0.28 will be released in 2021. Further "dot" releases of v0.27 may be published for security fixes in future.

The Libre Graphics Meeting was scheduled to take place in May 2020 in Rennes, France. I intended to conduct a workshop on **Image Metadata *and* Exiv2 Architecture**. This book was being written to be used in that presentation. Regretfully, the Covid-19 crisis caused the postponement of LGM.

I started working on Exiv2 to implement GeoTagging. As the years have passed, I've explored most of the code. I've added new capability such as support for ICC profiles, metadata-piping and file-debugging. I've done lots of work on the build, test suite and documentation. I've talked to users all over the world and closed several hundred issues and feature requests. Over the years, I've met users in India, Singapore, Armenia, the USA and the UK. I've attended 2 Open-Source Conferences. It's been an adventure and mostly rewarding. It's remarkable how seldom users express appreciation.

TOC

Current Development Priorities

In July 2017 we received our first security CVE. Not a pleasant experience. The security folks started hitting us with fuzzed files. These are files which violate format specifications and can cause the code to crash. We responded with v0.27 which will have regular "dot" releases to provide security fixes. Managing frequent releases and user correspondence consumes lots of my time.

In parallel with "the dots", major work is being carried to prepare Exiv2 for the future. Luis, Dan and Rosen are working on v0.28 which will be released in 2021. This is a considerable reworking of the code into C++11.

I'm delighted by the work done by Dan, Luis and Kevin to deal with the assault of the security people. I believe we are responding effectively to security issues. None-the-less, they have dominated the development of Exiv2 for at least two years and many ideas could not be pursued because security consumed our engineering resources.

TOC

Future Development Projects

The code is in good shape, our release process is solid and we have comprehensive user documentation. As photography develops, there will be many new cameras and more image formats such as CR3, HEIF and BigTiff. Exiv2 Video support is weak and was deprecated in v0.27. It will be removed in 0.28. One day a contributor will re-engineer the video code.

A long standing project for Exiv2 is a **unified metadata container**. There is an implementation of this in the SVN repository. Currently we have three containers for Exif, Iptc and Xmp. This is clumsy. We also have a restriction of one image per file. Perhaps both restrictions have a common solution.

The toolset used in Software Engineering evolves with time. C++ has been around for about 35 years and, while many complain about it, I expect it will out-live most of us. None-the-less, languages which are less vulnerable to security issues may lead the project to a re-write in a new language such as Rust. I hope this book provides the necessary understanding of metadata to support such an undertaking.

The most common issue raised by users concerns lens recognition. For v0.26, I added the ***Configuration File*** feature to enable users to modify lens recognition on their computer. While this is helpful, many users would like Exiv2 to deal with this perfectly, both now and in the future.

I intended to make a proposal at LGM in Rennes in May 2020 concerning this matter. Both Exiv2 and ExifTool can extract metadata from an image into a .EXV file. I would propose to implement a program to read the

.EXV and return the Lens. That program will have an embedded programming language with the rules to identify the lens. The scripts will be ascii files which can be updated. It will be called M2Lscript (MetaData to Lens Script), pronounced ***"MillsScript"***. The M2Lscript interpreter will be available as a command-line program, a perl module (for ExifTool), a C++ library (for linking into exiv2) and perhaps a python module.

In this way, new lens definitions can be written in M2Lscript without touching anything in Exiv2 or ExifTool.

I will not be able to work on both Exiv2 and M2Lscript simulateously. When a new maintainer takes responsibility for Exiv2, I will retire. M2Lscript would be my swansong technology project. However, the C-19 crisis postponed LGM in 2020. I don't have the energy to continue with open-source. This book is my final contribution.

TOC

Purpose and Scope of this book

This book is my gift and legacy to Exiv2. I hope Exiv2 will continue to exist long into the future. This book is being written to document my discoveries about **Image Metadata *and* Exiv2 Architecture**. However, I want to avoid a ***cut'n'paste*** of information already in the project documentation. This book is an effort to collect my knowledge of this code into a single volume. Many topics in this book are discussed in more detail in the issue history stored in Redmine and GitHub. I hope this book helps future maintainers to understand Exiv2, solve issues and develop the code for years to come.

I wish you a happy adventure in the world of Image Metadata. If you'd like to discuss matters concerning this book, please open an issue on GitHub and share your thoughts with Team Exiv2.

Disclaimer

Attention is drawn to the possibility that some elements of this document may be the subject of patent rights. Robin Mills and/or the Exiv2 Project and/or the Exiv2 Contributors shall not be held responsible for identifying any or all such patent rights.

TOC

Making this book

I've had a lot of fun making this book. Most of the time was spent on the code, however getting the book into good shape for the web and print has been fun. The graphics were drawn using OmniGraffle 6.6.2 on my MacBook Pro.

All the documentation for Exiv2 is written in markdown with the exception of the Unix man page exiv2.1 I find markdown easy to use and quickly produces satisfying results.

The book is written in markdown and displayed on my computer with the MacDown Application. When MacDown exports a PDF, he ignores print directives in the style sheet, he does not support page numbering and the links are ineffective. To my taste, the text size of pages is too large when printed on A4.

I used a modified version of this style sheet: ~/Library/Application Support/MacDown/Styles/GitHub2.css. I changed the fonts to be Helvetica in the titles and Palatino in the body. I thought about using the Exiv2 logo font which is Albertus Medium. I decided to adopt the ubiquitous Palatino. Code is set in Consolas in both the graphics and the in-line code snippets in the text.

```Bash
@media print {
    h1,h2 { page-break-before: always; }
    h3,h4 { page-break-after: never;   }
}
```

I get MacDown to export HTML to IMaEA.html. I open IMaEA.html in Safari and print it into a PDF file with a page size of 275x389mm. This preserves the aspect ratio √2/1 of ISO-Standard pages. Safari has a option to add page number and date to every page. I get Safari to save the print in PDF (it's 275x388). The printing system on macOS has a Paper Handling feature to scale the print to fit a page size and I set that to A4. I save the PDF from the print system and the result is a beautiful A4 document with all the links working and scaled to fit A4.

I have to manually update the page numbers in the table of contents. If Exiv2 ever supports PDF, I'll probably be able to script that! I only do that when I intend to publish the file as it's tedious.

The final step is to take the PDF to the local print shop to be printed and bound.

Incidentally, I investigated adding a clickable Exiv2 logo to every page of the PDF and found this very useful open-source program pdfstamp: url = https://github.com/CrossRef/pdfstamp.git

PDF documents work in point sizes (72/inch) so A4 pages 297x210mm = 842x596pt. The origin is in the lower left.

```Bash
$ java -jar pdfstamp.jar -v -i ~/gnu/exiv2/team/book/exiv2.png  -l 30,30 -u https:
$ java -jar pdfstamp.jar -v -d 8000 -i ~/gnu/exiv2/team/book/exiv2-large.png  -l 550,30
```

We could use this to add page labels (date/time/title) to every page (except the front page).

I also investigated doing this in the style-sheet. I tried Safari, Chrome and Firefox with varying success. Then I read this: https://www.smashingmagazine.com/2015/01/designing-for-print-with-css/.

The prince product fully supports HTML->PDF with @media print in the style sheet and works really well. They offer a free/unrestricted license for non-commercial use.

https://www.princexml.com

I tried prince and was very pleased with the result. When you ask prince to create the PDF, you can specify page-size and style sheet. I've set up IMaEA.css with the builtin page size of 275x389.

```Bash
$ prince --page-size='275mm 389mm' --style ~/gnu/exiv2/team/book/pdf-styles.css IM
$ prince --type IMaEA.css IMaEA.html
```

The date that appears at the center-bottom of every page (except the first) is in the style sheet. You could change that with sed of course. Setting the date from the computer clock would be fine for an automatic reporting application. Better to use static text as we might want to say "Exiv2 v0.27.3 2020-06-30" or the like.

The resulting PDF is beautiful and not watermarked by prince, although they put a postit on the front page. That's OK. They deserve credit for their outstanding work and free license.

However, prince rendered the code snippets as plain pre-formatted text and didn't provide the beautiful formatting and syntax colouring provide by MacDown and which is printed in the PDF generated by Safari.

So, I decided that the Safari/PDF was the best PDF and I tweaked the PDF in three ways using SodaPDF. I fixed the title and dates on every page. I fixed the "goto page#" PDF links which were mysteriously off by one page, and I added a PDF Table of Contents. The result is a beautiful document which looks great on the tablet (in HTML or PDF), great on the computer and beautiful when printed.

Thank You for reading my book. If you find errors, please let me know. If you'd like to discuss any of the technology involved in Image Metadata, please contact me by opening an issue on GitHub.
https://github.com/exiv2/exiv2

TOC

1 Image File Formats

The following summaries of the file formats are provided to help you to understand both this book and the Exiv2 code. The Standard Specifications should be consulted for more detail.

I've made a summary of every file format supported by Exiv2 and hope you find that useful. There are an absurd number of Graphics File Formats. I have a copy somewhere of the O'Reilly book. I got it in 1996 and it has 1000+ pages. Since then there have been many more invented. It's a software mess. In the early days, many formats were local to a few users in a University and escaped to a wider audience. However the never ending stream of new standards is horrible. Canon have several different RAW formats such as CRW, CR2 and CR3.

A good model for an image is to think of it as a container. It's like a directory on the disk. The directory can hold files with different formats and the directory is recursive as it can contain a directory of more files. Almost every graphics format since TIFF in 1992 is a container.

The good news however is that file formats come in families which are:

Family	**Description**	**Examples**
TIFF	You must learn Tiff thoroughly to understand metadata	TIFF, DNG, NEF, ICC, CR2, ORF, RAW, DCP,PEF
JIFF	JPEG Image File Format Linked list of 64k segments	JPEG, EXV
PNG	Another popular format Linked list of chunks	PNG
CIFF	Camera Image File Format. Dave Coffin parse.c decodes CRW	CRW
ISOBMFF	Based on the .mp4 format	MP4, CR3, AVI, HEIF, JP2
RIFF	Resource Interchange File Format	WEBP, AVI
GIF	Graphics Image Format	GIF
BMP	Windows BMP never has XMP, IPTC or Exif metadata. Version5 may include an ICC profile.	BMP
EPS	Adobe Encapsulated PostScript The code in Exiv2 to deal with this is deprecated	EPS, AI

The Metadata is defined by standards which also define how to embed the data in the image.

Standard	Description
Exif	EXchangeable Image Format. This is encoded as a TIFF sub-file
IPTC	International Press Telecommunications Council
ICC	Internation Colour Consortium The ICC Profile is similar to TIFF The ICC Profile is an ICC sub-file.
XMP	Adobe XMP is encoded as an XML sub-file

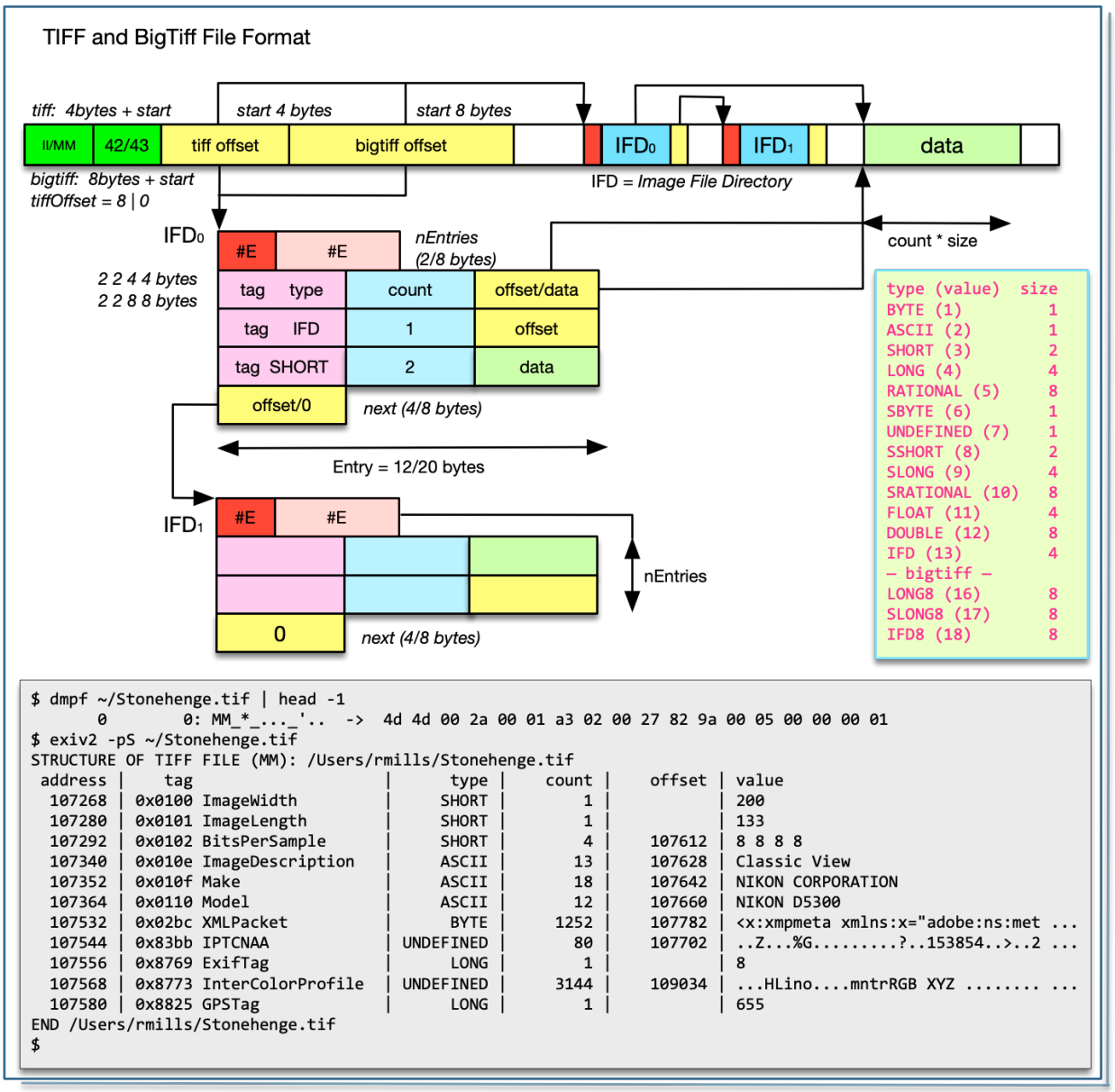
I suspect the proliferation of formats is caused by the hardware engineers. When hardware people start a new project, they copy the CAD files from the last project and proceed from there. They don't worry about back-porting changes or compatibility. We have to live with this mess.

There is also the issue of patents. It's unclear if it's legal to read an ISOBMFF file which is used by Apple to store Heif files. I believe it is legal to read ISOBMFF files. It's illegal to reverse engineer the proprietary encoded data stored in the mdat box a HEIF. Metadata is occasionally compressed (PNG), encrypted (Nikon) or ciphered (Sony).

Here is a useful WikiPedia site that summarises file formats:
https://en.wikipedia.org/wiki/Comparison_of_graphics_file_formats

TOC

Tagged Image File Format




```
$ dmpf ~/Stonehenge.tif | head -1
       0        0: MM_*_..._'..  ->  4d 4d 00 2a 00 01 a3 02 00 27 82 9a 00 05 00 00 00 01
$ exiv2 -pS ~/Stonehenge.tif
STRUCTURE OF TIFF FILE (MM): /Users/rmills/Stonehenge.tif
 address |    tag                      |      type |    count |    offset | value
  107268 | 0x0100 ImageWidth           |     SHORT |        1 |           | 200
  107280 | 0x0101 ImageLength          |     SHORT |        1 |           | 133
  107292 | 0x0102 BitsPerSample        |     SHORT |        4 |    107612 | 8 8 8 8
  107340 | 0x010e ImageDescription     |     ASCII |       13 |    107628 | Classic View
  107352 | 0x010f Make                 |     ASCII |       18 |    107642 | NIKON CORPORATION
  107364 | 0x0110 Model                |     ASCII |       12 |    107660 | NIKON D5300
  107532 | 0x02bc XMLPacket            |      BYTE |     1252 |    107782 | <x:xmpmeta xmlns:x="adobe:ns:met ...
  107544 | 0x83bb IPTCNAA              | UNDEFINED |       80 |    107702 | ..Z...%G.........?..153854..>..2 ...
  107556 | 0x8769 ExifTag              |      LONG |        1 |           | 8
  107568 | 0x8773 InterColorProfile    | UNDEFINED |     3144 |    109034 | ...HLino....mntrRGB XYZ ........ ...
  107580 | 0x8825 GPSTag               |      LONG |        1 |           | 655
END /Users/rmills/Stonehenge.tif
$
```

The architecture of TIFF and BigTiff are the same. BigTiff is 64 bit based. So most uint16_t data types become uint32_t and uint32_t become uint64_t. BigTiff has three additional 8 byte types: Long8, SLong8 and Ifd8.

For both TIFF and BigTiff, the ***magic*** header is MM (Motorola) for big-endian and II (Intel) for little-endian, followed by a 2-byte integer which must be 42 (ascii *) for Tiff and 43 (ascii +) for BigTiff. These markers are very obvious **MM_+** or **II*_** when formatted by dmpf.cpp

Both tag and type are uint16_t in TIFF and BigTiff.

The header for TIFF is 8 bytes. It is the ***magic*** header followed by a long offset to the first IFD. The header for BigTiff is 16 bytes. It is the ***magic*** header followed by 2 shorts (which must be 8,0) and a long8 offset to the first IFD.

Element	TIFF	BigTiff	Element	TIFF	BigTiff
Header	XX*_Long	XX+_ 8 0 Long8	Header	8 bytes	16 bytes
Marker	* 0x2a = 42	+ 0x2b = 43	Offset	uint32_t	uint64_t
Tag	uint16_t	uint16_t	Entry	12 bytes	20 bytes
Type	uint16_t	uint16_t	Entries # **E**	uint16_t	uint64_t
Count	uint32_t	uint64_t	Next	uint32_t	uint64_t

It's important to understand that Endian can change as we descend into the file. There are images in which there are sub-files whose endian setting is different from the container.

XMP and ICC Profiles in Tiff

These are defined in the following tags:

```Bash
$ taglist ALL | grep -e ^Image\.InterColorProfile  -e ^Image.XMLPacket | csv -
[Image.XMLPacket]   [700]   [0x02bc]    [Image] [Exif.Image.XMLPacket]  [Byte]  [XMP Met
[Image.InterColorProfile]   [34675] [0x8773]    [Image] [Exif.Image.InterColorProfile]
693 rmills@rmillsmm-local:~/gnu/exiv2/team/book $
$
```

NEF, DNG and CR2

These are tiff files. There must be some subtle matters to be handled in these formats, however tvisitor has no trouble running over the files. Allow me to quote directly from Adobe's document: https://wwwimages2.adobe.com/content/dam/acom/en/products/photoshop/pdfs/dngspec1.5.0.0.pdf

A Standard Format

The lack of a standard format for camera raw files creates additional work for camera manufacturers because they need to develop proprietary formats along with the software to process them. It also poses risks for end users. Camera raw formats vary from camera to camera, even those produced by the same manufacturer. It is not uncommon for a camera manufacturer to terminate support for a discontinued camera's raw format. This means users have no guarantee they will be able to open archived camera raw files in the future.

To address these problems, Adobe has defined a new non-proprietary format for camera raw files. The format, called Digital Negative or DNG, can be used by a wide range of hardware and software developers to provide a more flexible raw processing and archiving workflow. End users may also use DNG as an intermediate format for storing images that were originally captured using a proprietary camera raw format.

TIFF Compatible

DNG is an extension of the TIFF 6.0 format, and is compatible with the TIFF-EP standard. It is possible (but not required) for a DNG file to simultaneously comply with both the Digital Negative specification and the TIFF-EP standard.

I downloaded and installed Adobe's DNG Convertor and applied it to some NEF files from my Nikon D5300:

```Bash
.../book/build $ ./tvisitor -pU /Users/rmills/temp/Raw/DSC_0003.dng
STRUCTURE OF TIFF FILE (II): /Users/rmills/temp/Raw/DSC_0003.dng
 address |    tag                              |      type |    count |    offset | valu
      10 | 0x00fe Exif.Image.NewSubfileType    |      LONG |        1 |           | 1
      22 | 0x0100 Exif.Image.ImageWidth        |      LONG |        1 |           | 256
      34 | 0x0101 Exif.Image.ImageLength       |      LONG |        1 |           | 171
      46 | 0x0102 Exif.Image.BitsPerSample     |     SHORT |        3 |       734 | 8 8
      58 | 0x0103 Exif.Image.Compression       |     SHORT |        1 |           | 1
      70 | 0x0106 Exif.Image.PhotometricInte.. |     SHORT |        1 |           | 2
      82 | 0x010f Exif.Image.Make              |     ASCII |       18 |       740 | NIK(
      94 | 0x0110 Exif.Image.Model             |     ASCII |       12 |       758 | NIK(
     106 | 0x0111 Exif.Image.StripOffsets      |      LONG |        1 |           | 286
     118 | 0x0112 Exif.Image.Orientation       |     SHORT |        1 |           | 1
     130 | 0x0115 Exif.Image.SamplesPerPixel   |     SHORT |        1 |           | 3
     142 | 0x0116 Exif.Image.RowsPerStrip      |      LONG |        1 |           | 171
     154 | 0x0117 Exif.Image.StripByteCounts   |      LONG |        1 |           | 131
     166 | 0x011c Exif.Image.PlanarConfigura.. |     SHORT |        1 |           | 1
     178 | 0x0131 Exif.Image.Software          |     ASCII |       37 |       770 | Adol
     190 | 0x0132 Exif.Image.DateTime          |     ASCII |       20 |       808 | 202(
     202 | 0x014a Exif.Image.SubIFD            |      LONG |        2 |       828 | 280(
  STRUCTURE OF TIFF FILE (II): /Users/rmills/temp/Raw/DSC_0003.dng
   address |    tag                              |      type |    count |    offset | v
    280024 | 0x00fe Exif.Image.NewSubfileType    |      LONG |        1 |           | 0
    280036 | 0x0100 Exif.Image.ImageWidth        |      LONG |        1 |           | 6
    280048 | 0x0101 Exif.Image.ImageLength       |      LONG |        1 |           | 4
    280060 | 0x0102 Exif.Image.BitsPerSample     |     SHORT |        1 |           | 1
    280072 | 0x0103 Exif.Image.Compression       |     SHORT |        1 |           | 7
    280084 | 0x0106 Exif.Image.PhotometricInte.. |     SHORT |        1 |           | 3
    280096 | 0x0115 Exif.Image.SamplesPerPixel   |     SHORT |        1 |           | 1
    280108 | 0x011c Exif.Image.PlanarConfigura.. |     SHORT |        1 |           | 1
    280120 | 0x0142 Exif.Image.0x142             |      LONG |        1 |           | 2
    280132 | 0x0143 Exif.Image.0x143             |      LONG |        1 |           | 2
...
    280348 | 0xc761 Exif.Image.0xc761            |    DOUBLE |        6 |    285092 | 4
  END: /Users/rmills/temp/Raw/DSC_0003.dng
...
     706 | 0xc761 Exif.Image.0xc761            |    DOUBLE |        6 |    279958 | 454
     718 | 0xc7a7 Exif.Image.0xc7a7            |     UBYTE |       16 |    280006 | 51
END: /Users/rmills/temp/Raw/DSC_0003.dng
```

I was a little surprised that Adobe have removed the MakerNote.

I believe the "undefined" tags which are listed in the format: Exif.Image.0xc761 and defined in the specification. C761.H is "Noise Profile" for which the mathematics are explained by Adobe!

CR2 and NEF will require more investigation.

It's possible that there are tags which are unique to CR2 and NEF and tvisitor.cpp is hiding them when the *U* option is not being used. In the first instance, I can search in the Exiv2 source code to see if there is any special or unusual being defined or used by the CR2 and NEF handlers.

Garbage Collecting Tiff Files

There is a significant problem with the Tiff format. It's possible for binary records to hold offsets to significant data elsewhere in the file. This creates two problems. Firstly, when buried in an undocumented MakerNote, we don't know that the data is an offset. So, when all the blocks move in a rewrite of the file, we can neither relocate the referenced data, nor update the offset. My conclusion is that is almost impossible to garbage collect a tiff file. However, the situation isn't hopeless. The offset in the Tiff Header defines the location of IFD0. It's very common that IFD0 is at the end of the file and the reason is obvious. When a Tiff is rewritten by an application, they create IFD0 in memory, then copy it to the end of the file and update the offset in the header. If we are creating IFD0, we can safely reuse the spaced occuped by previous IFD0.

Imperial College have medical imaging Tiff files which are of the order of 100 GigaBytes in length. Clearly we do not want to rewrite such a file to modify a few bytes of metadata. We determine the new IFD0 and write it at end of the file.

When we update a Makernote, we should "edit in place" and always avoid relocating the data. Regrettably for a JPEG, that's almost impossible. As camera manufacturers have higher resolutions and larger displays for review, the manufacturers want to have larger thumbnails and are happy to store the preview somewhere in the JPEG and have a hidden offset in the makernote. This works fine until the image is edited when the preview is lost.

In principle, a Tiff can be garbage collected with a block-map. If we set up a block-map with one bit for every thousand bytes, we can run the IFDs and mark all the blocks in use. When we rewrite the TIFF (well IFD0 actually), we can inspect the block-map to determine a "hole" in the file at which to write. I would not do this. It's unsafe to over-write anything in a Tiff with the exception of IFD0 and the header offset. The situation with JPEG is more serious. It's impossible to rewrite the JPEG in place.

The concept of using a block-map to track known data is used in RemoteIo. We use a block-map to avoid excessive remote I/O by reading data into a cache. We never read data twice. We do not need contiguous memory for the file. This is discussed in 5. I/O in Exiv2

I would like to express my dismay with the design of most image containers. There is a much simpler design used by macOS and that is a bundle. A bundle is a directory of files which includes the file Info.plist. It appears in the Finder to be a simple entity like a file. The terminal command ***ditto*** is provided to copy them. All programming languages can manipulte files. The metadata in an image should be a little Tiff or sidecar in a bundle. In principle, a container such as Tiff is a collection of streams that are both relocatable and never reference external data. Sadly, TIFF and JPEG make it very easy to break both rules. The design of JPEG makes it almost impossible to edit anything without relocating all the data. The situation with video is even more serious as the files are huge. In the PDF format, the file maintains a directory of objects. The objects can be safely relocated because objects reference each other by name and not the file offset.

Metadata that cannot be edited

There are tags in Tiff such as ***ImageWidth*** which cannot be modified without rewriting the pixels in the image. Exif protects those tags in the functions **TiffHeader::isImageTag()** and **Cr2Header::isImageTag()**.

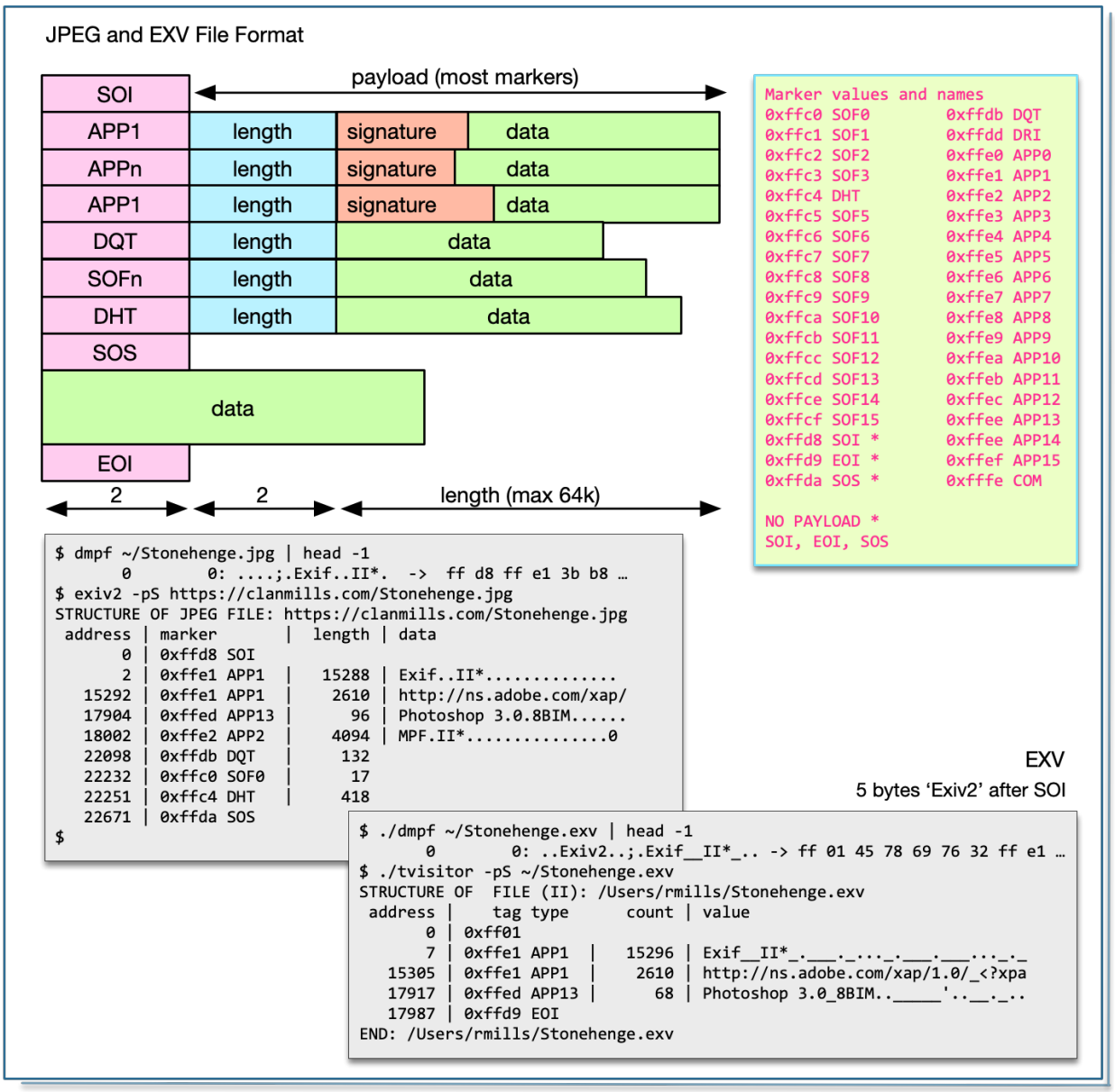
DCP Camera Profiles

The Adobe Camera Raw Convertor installs CameraProfiles .dcp files in /Library/Application Support/Adobe/CameraRaw/CameraProfiles/ (on macOS). Camera Profiles are defined in the Adobe DNG Specification. They are a modified TIFF format which has the Signature "IIRClong". Example:

```Bash
569 rmills@rmillsmm-local:~/gnu/exiv2/team/book $ dmpf count=40 files/NikonD5300.d
       0        0: IIRC.___._...._.___.___!.._._.___._  ->  49 49 52 43 08 00 00 00 11 00 1
    0x20       32: __"..._._                              ->  00 00 22 c6 0a 00 09 00
570 rmills@rmillsmm-local:~/gnu/exiv2/team/book $ build/tvisitor files/NikonD5300.dcp
STRUCTURE OF TIFF FILE (II): files/NikonD5300.dcp
 address |    tag                                |      type |    count |    offset | valu
      10 | 0xc614 Exif.DNG.UniqueCameraModel     |     ASCII |       12 |       218 | Nik
      22 | 0xc621 Exif.DNG.ColorMatrix1          | SRATIONAL |        9 |       230 | 967
      34 | 0xc622 Exif.DNG.ColorMatrix2          | SRATIONAL |        9 |       302 | 698
      46 | 0xc65a Exif.DNG.CalibrationIllumi..   |     SHORT |        1 |           | 17
      58 | 0xc65b Exif.DNG.CalibrationIllumi..   |     SHORT |        1 |           | 21
      70 | 0xc6f4 Exif.DNG.ProfileCalibratio..   |     ASCII |       10 |       374 | com
      82 | 0xc6f8 Exif.DNG.ProfileName           |     ASCII |       17 |       384 | Came
      94 | 0xc6fc Exif.DNG.ProfileToneCurve      |     FLOAT |      128 |       402 | 0 0
     106 | 0xc6fd Exif.DNG.ProfileEmbedPolicy    |      LONG |        1 |           | 1
     118 | 0xc6fe Exif.DNG.ProfileCopyright      |     ASCII |       35 |       914 | Copy
     130 | 0xc714 Exif.DNG.ForwardMatrix1        | SRATIONAL |        9 |       950 | 797
     142 | 0xc715 Exif.DNG.ForwardMatrix2        | SRATIONAL |        9 |      1022 | 797
     154 | 0xc725 Exif.DNG.ProfileLookTableD..   |      LONG |        3 |      1094 | 90 1
     166 | 0xc726 Exif.DNG.ProfileLookTableD..   |     FLOAT |    69120 |      1106 | 0 10
     178 | 0xc7a4 Exif.DNG.ProfileLookTableE..   |      LONG |        1 |           | 1
     190 | 0xc7a5 Exif.DNG.BaselineExposureO..   | SRATIONAL |        1 |    277586 | 429
     202 | 0xc7a6 Exif.DNG.DefaultBlackRender    |      LONG |        1 |           | 1
END: files/NikonD5300.dcp
571 rmills@rmillsmm-local:~/gnu/exiv2/team/book $
```

TOC

JPEG and EXV Format



JPEG and EXF are almost the same thing, however most graphics applications will reject EXF because it is not a valid JPEG. ExifTool also supports EXF. In tvisitor.cpp, class JpegImage handles both and the only difference is respected in JpegImage::valid():

```C++
bool JpegImage::valid()
{
    IoSave   restore(io(),0);
    bool     result = false;
    byte     h[2];
    io_.read(h,2);
    if ( h[0] == 0xff && h[1] == 0xd8 ) { // .JPEG
        start_  = 0;
        format_ = "JPEG";
        endian_ = keLittle;
        result  = true;
    } else if  ( h[0] == 0xff && h[1]==0x01 ) { // .EXV
        DataBuf buf(5);
        io_.read(buf);
        if ( buf.is("Exiv2") ) {
            start_ = 7;
            format_ = "EXV";
            endian_ = keLittle;
            result = true;
        }
    }
    return result;
} // JpegImage::valid()
```

And here it is in action:

```Bash
.../book/build $ ./tvisitor -pS ~/Stonehenge.jpg
STRUCTURE OF JPEG FILE (II): /Users/rmills/Stonehenge.jpg
 address |    tag type      count | value
       0 | 0xffd8 SOI
       2 | 0xffe1 APP1  |   15288 | Exif__II*_.___._...._.___.___...._.___ +++
   15292 | 0xffe1 APP1  |    2610 | http://ns.adobe.com/xap/1.0/_<?xpack +++
   17904 | 0xffed APP13 |      96 | Photoshop 3.0_8BIM.._____'..__._...Z +++
   18002 | 0xffe2 APP2  |    4094 | MPF_II*_.___.__..._.___0100..._.___._ +++
   22098 | 0xffdb DQT   |     132
   22232 | 0xffc0 SOF0  |      17
   22251 | 0xffc4 DHT   |     418
   22671 | 0xffda SOS
END: /Users/rmills/Stonehenge.jpg
.../book/build $ exiv2 -ea --verbose --force ~/Stonehenge.jpg
File 1/1: /Users/rmills/Stonehenge.jpg
Writing Exif data from /Users/rmills/Stonehenge.jpg to /Users/rmills/Stonehenge.exv
Writing IPTC data from /Users/rmills/Stonehenge.jpg to /Users/rmills/Stonehenge.exv
Writing XMP data from /Users/rmills/Stonehenge.jpg to /Users/rmills/Stonehenge.exv
.../book/build $ ./tvisitor -pS ~/Stonehenge.exv
STRUCTURE OF EXV FILE (II): /Users/rmills/Stonehenge.exv
 address |    tag type      count | value
       0 | 0xff01
       7 | 0xffe1 APP1  |   15296 | Exif__II*_.___._...._.___.___...._.___ +++
   15305 | 0xffe1 APP1  |    2610 | http://ns.adobe.com/xap/1.0/_<?xpack +++
   17917 | 0xffed APP13 |      68 | Photoshop 3.0_8BIM.._____..__._...Z +++
   17987 | 0xffd9 EOI
END: /Users/rmills/Stonehenge.exv
```

```
.../book/build $
```

APP13 Photoshop 3.0 Segment

This is an 8BIM chain and is explained in PSD PhotoShop Document

Extended JPEG

The JPEG standard restricts a single segment of a JPEG to 64k bytes because the length field is a 16 bit uint16_t. Exif, XMP and ICC frequently exceed 64k. Regrettably three different schemes are used to enable multiple consecutive segments to be coalesced into a larger block.

tvisitor.cpp supports Adobe and AGFA extended JPEG.

Adobe Exif >64k in JPEG

Adobe have created an ***ad-hoc*** standard by placing consecutive APP1 segments with the signature **Exif\0\0**. This ***ad-hoc*** standard is defined in Adobe's XMP Specification Part 3 2016+.

Exiv2 has no code to deal with this. It can neither read nor write these files. In fact, JpegImage::writeMetadata() currently throws when asked to write more than 64k into a JPEG.

This is discussed here: https://dev.exiv2.org/issues/1232 and here is the output of the test files which were contributed by Phil.

Bash

```
.../book/build $ ./tvisitor -pS ~/cs4_extended_exif.jpg
STRUCTURE OF JPEG FILE (II): /Users/rmills/cs4_extended_exif.jpg
 address | marker       |  length | signature
       0 | 0xffd8 SOI
       2 | 0xffe0 APP0  |      16 | JFIF_..._._.__..
      20 | 0xffe1 APP1  |   65498 | Exif__MM_*___._..._.___.___n.._.___._.__
   65520 | 0xffe1 APP1  |   65498 | Exif__g keys we require'd            nex
  131020 | 0xffe1 APP1  |   52820 | Exif__)            if ($segDataPt =~ /^
  183842 | 0xffed APP13 |    4440 | Photoshop 3.0_8BIM.._____...__.__..x..#-
  188284 | 0xffe1 APP1  |    4323 | http://ns.adobe.com/xap/1.0/_<?xpacket b
  192609 | 0xffe1 APP1  |   65477 | http://ns.adobe.com/xmp/extension/_C8400
  258088 | 0xffe1 APP1  |   65477 | http://ns.adobe.com/xmp/extension/_C8400
  323567 | 0xffe1 APP1  |   56466 | http://ns.adobe.com/xmp/extension/_C8400
  380035 | 0xffe2 APP2  |    3160 | ICC_PROFILE_..__.HLino..__mntrRGB XYZ ..
  383197 | 0xffee APP14 |      14 | Adobe_d.___...
  383213 | 0xffdb DQT   |     132 | _.......................................
  383347 | 0xffc0 SOF0  |      17 | ..T...."_........
  383366 | 0xffdd DRI   |       4 | _...
  383372 | 0xffc4 DHT   |     319 | __........_______._..........._........_
  383693 | 0xffda SOS   |      12 | .._....._?_.T
END: /Users/rmills/cs4_extended_exif.jpg
.../book/build $ ./tvisitor -pS ~/multi-segment_exif.jpg
STRUCTURE OF JPEG FILE (II): /Users/rmills/multi-segment_exif.jpg
 address | marker       |  length | signature
       0 | 0xffd8 SOI
       2 | 0xffe1 APP1  |   65535 | Exif__II*_.___._..._.___.___..._.___.___
   65539 | 0xffe1 APP1  |    5603 | Exif__.................................
```

```
   71144 | 0xffdb DQT   |     132 | _.....................................
   71278 | 0xffc4 DHT   |     418 | __........_________.............._.........
   71698 | 0xffc0 SOF0  |      17 | ..0.@..!_........
   71717 | 0xffda SOS   |      12 | .._....._?_..
END: /Users/rmills/multi-segment_exif.jpg
.../book/build $
```

AGFA Exif >64k in JPEG

This is discussed in https://dev.exiv2.org/issues/1232 I think it is desirable to support reading this data. Exiv2 should write using Adobe's JPEG > 64k ***ad-hoc*** standard.

```
.../book/build $ ./tvisitor -pS ~/Agfa.jpg
STRUCTURE OF JPEG FILE (II): /Users/rmills/Agfa.jpg
 address | marker       |  length | signature
       0 | 0xffd8 SOI
       2 | 0xffe1 APP1  |   46459 | Exif__II*_.___._..._.____.___..._.___.___..._._
   46463 | 0xffe3 APP3  |   65535 | ...._._.....................................
  112000 | 0xffe4 APP4  |   65535 | ..Hc..w .8<...z..M.77.h...{......C.y1...... .k
  177537 | 0xffe5 APP5  |    7243 | .U......K..u=).pl.W.F...B.$.3....mg}q.....Hb.m
  184782 | 0xffdb DQT   |     132 |
  184916 | 0xffc0 SOF0  |      17 |
  184935 | 0xffc4 DHT   |      75 |
  185012 | 0xffda SOS   |      12 |
END: /Users/rmills/Agfa.jpg
.../book/build $
```

The Agfa MakerNote contains an IFD which is preceded by **ABC_II#E** where #E is number of entries in the IFD. This is discussed in 2.5 MakerNotes

ICC Profile data > 64k in JPEG

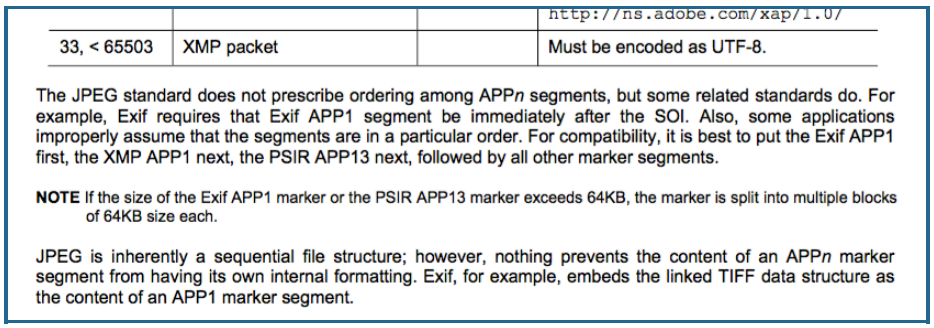
This is documented by ICC in ICC1v43_2010-12.pdf and implemented in Exiv2 for both reading and writing. The ICC profile has a signature of ICC_PROFILE_ followed by two uint8_t values which are the chunk sequence and the chunks count. The remainder of the data is the ICC profile. The test file test/data/ReaganLargeJpg.jpg has data in the format.

```
1155 rmills@rmillsmbp:~/gnu/github/exiv2/0.27-maintenance $ exiv2 -pS test/data/Re
STRUCTURE OF JPEG FILE: test/data/ReaganLargeJpg.jpg
 address | marker       |  length | data
       0 | 0xffd8 SOI
       2 | 0xffe0 APP0  |      16 | JFIF.....,.,....
      20 | 0xffe1 APP1  |    4073 | Exif..MM.*.....................
    4095 | 0xffe1 APP1  |    6191 | http://ns.adobe.com/xap/1.0/.<?x
   10288 | 0xffe2 APP2  |   65535 | ICC_PROFILE..... APPL....prtrRG chunk 1/25
   75825 | 0xffe2 APP2  |   65535 | ICC_PROFILE....S...r.R...t.RT..w chunk 2/25
  141362 | 0xffe2 APP2  |   65535 | ICC_PROFILE.....o..b.tn..Q.Km... chunk 3/25
...
 1517639 | 0xffe2 APP2  |   65535 | ICC_PROFILE...9.0.894.0.901.0.90 chunk 24/25
 1583176 | 0xffe2 APP2  |   41160 | ICC_PROFILE....463.0.465.0.469.0 chunk 25/25
 1624338 | 0xffdb DQT   |      67
 1624407 | 0xffdb DQT   |      67
 1624476 | 0xffc2 SOF2  |      17
 1624495 | 0xffc4 DHT   |      30
 1624527 | 0xffc4 DHT   |      27
 1624556 | 0xffda SOS
1156 rmills@rmillsmbp:~/gnu/github/exiv2/0.27-maintenance $
```

Bash

XMP data > 64k in JPEG

This is documented by Adobe in the XMP Specification 2016+ and implemented in Exiv2 in the API **JpegBase::printStructure::(kpsXMP)**. It is not implemented in **JpegBase::readMetadata()**.



			http://ns.adobe.com/xap/1.0/
33, < 65503	XMP packet		Must be encoded as UTF-8.

The JPEG standard does not prescribe ordering among APP*n* segments, but some related standards do. For example, Exif requires that Exif APP1 segment be immediately after the SOI. Also, some applications improperly assume that the segments are in a particular order. For compatibility, it is best to put the Exif APP1 first, the XMP APP1 next, the PSIR APP13 next, followed by all other marker segments.

NOTE If the size of the Exif APP1 marker or the PSIR APP13 marker exceeds 64KB, the marker is split into multiple blocks of 64KB size each.

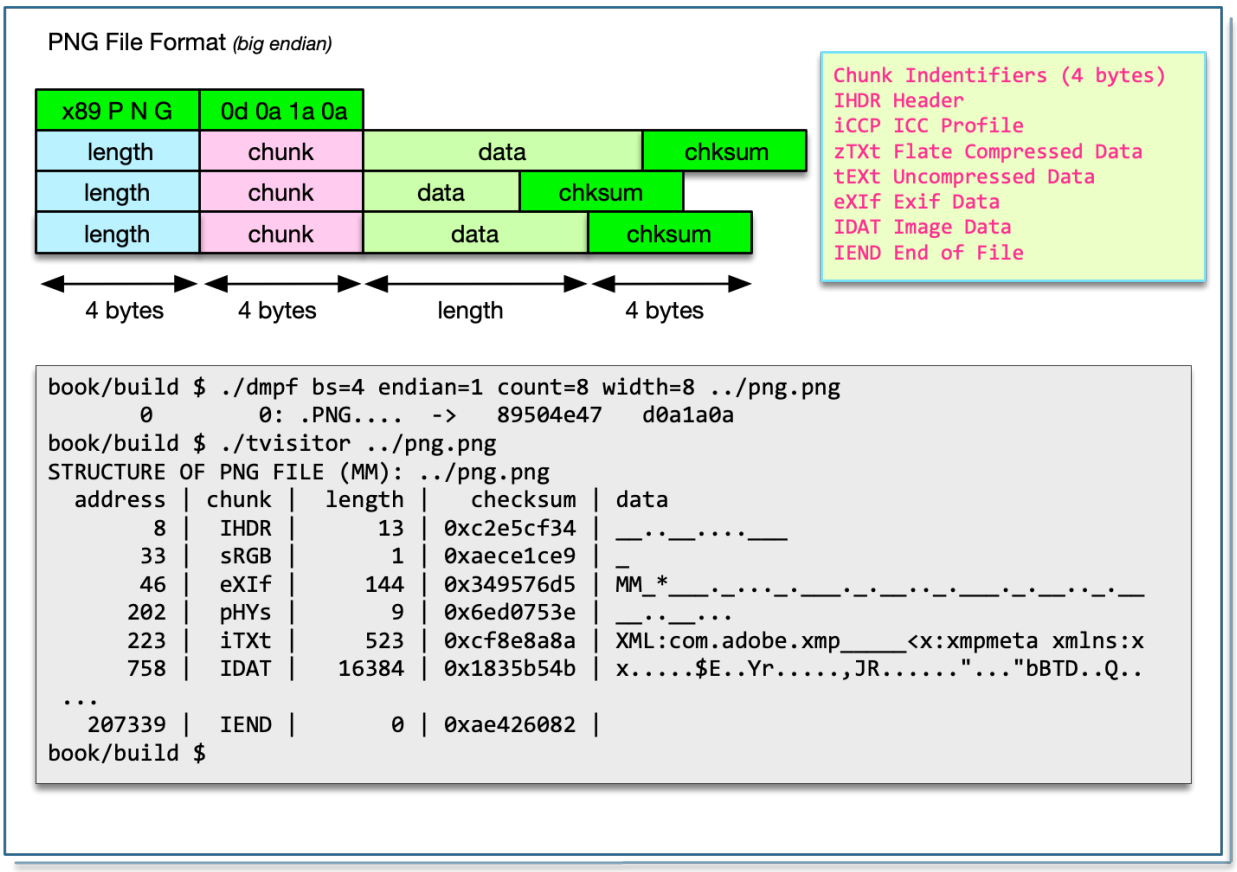
JPEG is inherently a sequential file structure; however, nothing prevents the content of an APP*n* marker segment from having its own internal formatting. Exif, for example, embeds the linked TIFF data structure as the content of an APP1 marker segment.

Other Unusual Adobe JPEG Features

Adobe have implemented transparency in JPEG by storing a PostScript clippath in the APP13 Photoshop 3.0 segment. Exiv2 has no code to deal with this. There is an Exif tag ClipPath which Exiv2 does support. I have encountered PhotoShop APP13 transparency. I've never encountered Exif.Image.ClipPath.

TOC

PNG Portable Network Graphics




```
book/build $ ./dmpf bs=4 endian=1 count=8 width=8 ../png.png
       0        0: .PNG....  ->   89504e47   d0a1a0a
book/build $ ./tvisitor ../png.png
STRUCTURE OF PNG FILE (MM): ../png.png
  address | chunk |  length |   checksum | data
        8 |  IHDR |      13 | 0xc2e5cf34 | __..__....___
       33 |  sRGB |       1 | 0xaece1ce9 | _
       46 |  eXIf |     144 | 0x349576d5 | MM_*___._...._.___._._.__..._.___._._.__..._.__
      202 |  pHYs |       9 | 0x6ed0753e | __..__...
      223 |  iTXt |     523 | 0xcf8e8a8a | XML:com.adobe.xmp_____<x:xmpmeta xmlns:x
      758 |  IDAT |   16384 | 0x1835b54b | x.....$E..Yr.....,JR......"..."bBTD..Q..
 ...
   207339 |  IEND |       0 | 0xae426082 |
book/build $
```

The PNG specification is available https://www.w3.org/TR/2003/REC-PNG-20031110/.

PNG is always bigEndian encoded. PNG has an 8 byte fixed header followed by a linked list of chunks. A chunk is 12 or more bytes and has a uint32_t length, char[4] chunk identifier, followed by binary data. The chunk data is trailed by a uint32_t checksum calculated by the zlib compression library.

We validate a PNG with the following code:

```C++
bool PngImage::valid()
{
    IoSave   restore(io(),0);
    bool     result  = true ;
    const byte pngHeader[] = { 0x89, 0x50, 0x4E, 0x47, 0x0D, 0x0A, 0x1A, 0x0A };
    for ( size_t i = 0 ; result && i < sizeof (pngHeader ); i ++) {
        result = io().getb() == pngHeader[i];
    }
    if ( result ) {
        start_  = 8       ;
        endian_ = keBig   ;
        format_ = "PNG"   ;
    }
    return result;
}
```

Navigating a PNG is straight forward:

```C++
void PngImage::accept(class Visitor& v)
{
    if ( valid() ) {
        v.visitBegin(*this);
        IoSave restore(io(),start_);
        uint64_t address = start_ ;
        while (  address < io().size() ) {
            io().seek(address );
            uint32_t  length  = io().getLong(endian_);
            uint64_t  next    = address + length + 12;
            char      chunk  [5] ;
            io().read(chunk  ,4) ;
            chunk[4]        = 0  ; // nul byte

            io().seek(next-4);                              // jump over data to checksum
            uint32_t  chksum  = io().getLong(endian_);
            v.visitChunk(io(),*this,address,chunk,length,chksum); // tell the visitor
            address = next ;
        }
        v.visitEnd(*this);
    }
}
```

Reporting Exif and XMP is also easy.

```C++
void Visitor::visitChunk(Io& io,Image& image
        ,uint64_t address,char* chunk,uint32_t length,uint32_t chksum)
{
    IoSave save(io,address+8);
    DataBuf   data(length);
    io.read(data);

    if ( option() & (kpsBasic | kpsRecursive) ) {
        out() << stringFormat(" %8d |  %s | %7d | %#10x | ",address,chunk,length,chksum)
        if ( length > 40 ) length = 40;
        out() << data.toString(kttUndefined,length,image.endian()) << std::endl;
    }

    if ( option() & kpsRecursive && std::strcmp(chunk,"eXIf") == 0 ) {
        Io        tiff(io,address+8,length);
        TiffImage(tiff).accept(*this);
    }

    if ( option() & kpsXMP && std::strcmp(chunk,"iTXt")==0 ) {
        if ( data.strcmp("XML:com.adobe.xmp")==0 ) {
            out() << data.pData_+22 ;
        }
    }
}
```

PNG ICC Profiles and XMP

As PNG chunks have a 32 bit length field, they can be stored as a single chunk. We don't need the messy arrangements used in JPEG to distribute data into multiple segments of less than 64k. XMP is normally stored as a iTXt/uncompressed or zTXt/compressed block. The signature at the start of the chunk is never compressed.

When an ICC profile is required, it is stored as an iCCP chunk. The signature is "ICC Profile". The profile is always compressed.

```Bash
1174 rmills@rmillsmbp:~/gnu/github/exiv2/0.27-maintenance $ exiv2 -pS test/data/Re
STRUCTURE OF PNG FILE: test/data/ReaganLargePng.png
 address | chunk |  length | data                              | checksum
       8 | IHDR  |      13 | .............                     | 0x8cf910c3
      33 | zTXt  |    8461 | Raw profile type exif..x...iv.    | 0x91fbf6a0
    8506 | zTXt  |     636 | Raw profile type iptc..x..TKn.    | 0x4e5178d3
    9154 | iTXt  |    7156 | XML:com.adobe.xmp.....<?xpacke    | 0x8d6d70ba
   16322 | gAMA  |       4 | ....                              | 0x0bfc6105
   16338 | iCCP  | 1151535 | ICC profile__x...UP.........!!    | 0x11f49e31
 1167885 | bKGD  |       6 | ......                            | 0xa0bda793
 1167903 | pHYs  |       9 | ...#...#.                         | 0x78a53f76
 1167924 | tIME  |       7 | ......2                           | 0x582d32e4
 1167943 | zTXt  |     278 | Comment..x.}..n.@....O..5..h..    | 0xdb1dfff5
 1168233 | IDAT  |    8192 | x...k.%.u%....D......GWW...ER.    | 0x929ed75c
 1176437 | IDAT  |    8192 | .F('.T)\....D"]..."2 '(...D%..    | 0x52c572c0
 1184641 | IDAT  |    8192 | y-.....>....3..p.....$....E.Bj    | 0x65a90ffb
 1192845 | IDAT  |    8192 | ....S....?..G.....G.........      | 0xf44da161
 1201049 | IDAT  |    7173 | .evl...3K..j.S.....x......Z .D    | 0xbe6d3574
 1208234 | IEND  |       0 |                                   | 0xae426082
1175 rmills@rmillsmbp:~/gnu/github/exiv2/0.27-maintenance $
```

PNG and the Zlib compression library

Some PNG chunks are flate compressed (lossless). You can build tvisitor.cpp with/without the Zlib compression flag using the cmake option -DEXIV2_ENABLE_PNG. This becomes the compiler define HAVE_LIBZ which enables additional code.

I'm very pleased to say that neither the Exiv2 or XMP metadata in the image book/png.png have been compressed and can be easily reported by tvisitor.cpp. It's very satisfying to use images from this book as test data for the code in this book.

Several chunks are always compressed. For example, zTXt and iCCP. The payload of zTXt normally comprises a nul-terminated signature, a one byte compression flag (always zero) followed by compressed data. For example:

```Bash
address | chunk |  length |   checksum | data
  16338 |  iCCP | 1151535 | 0x11f49e31 | ICC profile__x...UP........_.!!B....a.qwW | _.
                                        <signature>__compressed data             = dec
```

The flate compressed ICC profile follows the "ICC Profile__" signature.

The signatures: "Raw profile type iptc" and "Raw profile type exif" introduce a compressed block which when expanded is an ascii string with the following format. The number is count of hex code bytes. This is

redundant and does not need to be decoded.

```
\n
exif\n
    number\n
hexEncodedBinary\n
....
```

This data is revealed by tvisitor as follows:

```
 addr | chunk | length |   checksum | data                              | decompresse
   33 |  zTXt | 8461   | 0x91fbf6a0 | Raw profile type exif__x...iv.    | .exif.    8414.4
 8506 |  zTXt |  636   | 0x4e5178d3 | Raw profile type iptc__x..TKn.    | .iptc.     778.3
```

Converting hex encoded binary is straight-forward. Because hex encoded binary is always longer than the data, decoding updates the input buffer and returns the updated length.

```C++
static int hexToString(char buff[],int length)
{
    int  r     = 0   ; // resulting length
    int  t     = 0   ; // temporary
    bool first = true;
    bool valid[256];
    int  value[256];
    for ( int i =  0  ; i < 256 ; i++ ) valid[i] = false;
    for ( int i = '0' ; i <= '9' ; i++ ) {
        valid[i] = true;
        value[i] = i - '0';
    }
    for ( int i = 'a' ; i <= 'f' ; i++ ) {
        valid[i] = true;
        value[i] = 10 + i - 'a';
    }
    for ( int i = 'A' ; i <= 'F' ; i++ ) {
        valid[i] = true;
        value[i] = 10 + i - 'A';
    }

    for (int i = 0; i < length ; i++ )
    {
        char x = buff[i];
        if ( valid[x]  ) {
            if ( first ) {
                t     = value[x] << 4;
                first = false ;
            } else  {
                first  = true;
                buff[r++] = t + value[x];
            }
        }
    }
    return r;
}
```

Here is an important section of the standard concerning textual metadata:

11.3.4 Textual information

11.3.4.1 Introduction

PNG provides the tEXt, iTXt, and zTXt chunks for storing text strings associated with the image, such as an image description or copyright notice. Keywords are used to indicate what each text string represents. Any number of such text chunks may appear, and more than one with the same keyword is permitted.

11.3.4.2 Keywords and text strings

The following keywords are predefined and should be used where appropriate.

Title	Short (one line) title or caption for image
Author	Name of image's creator
Description	Description of image (possibly long)
Copyright	Copyright notice
Creation Time	Time of original image creation
Software	Software used to create the image
Disclaimer	Legal disclaimer
Warning	Warning of nature of content
Source	Device used to create the image
Comment	Miscellaneous comment

Other keywords may be defined for other purposes. Keywords of general interest can be registered with the PNG Registration Authority. It is also permitted to use private unregistered keywords.

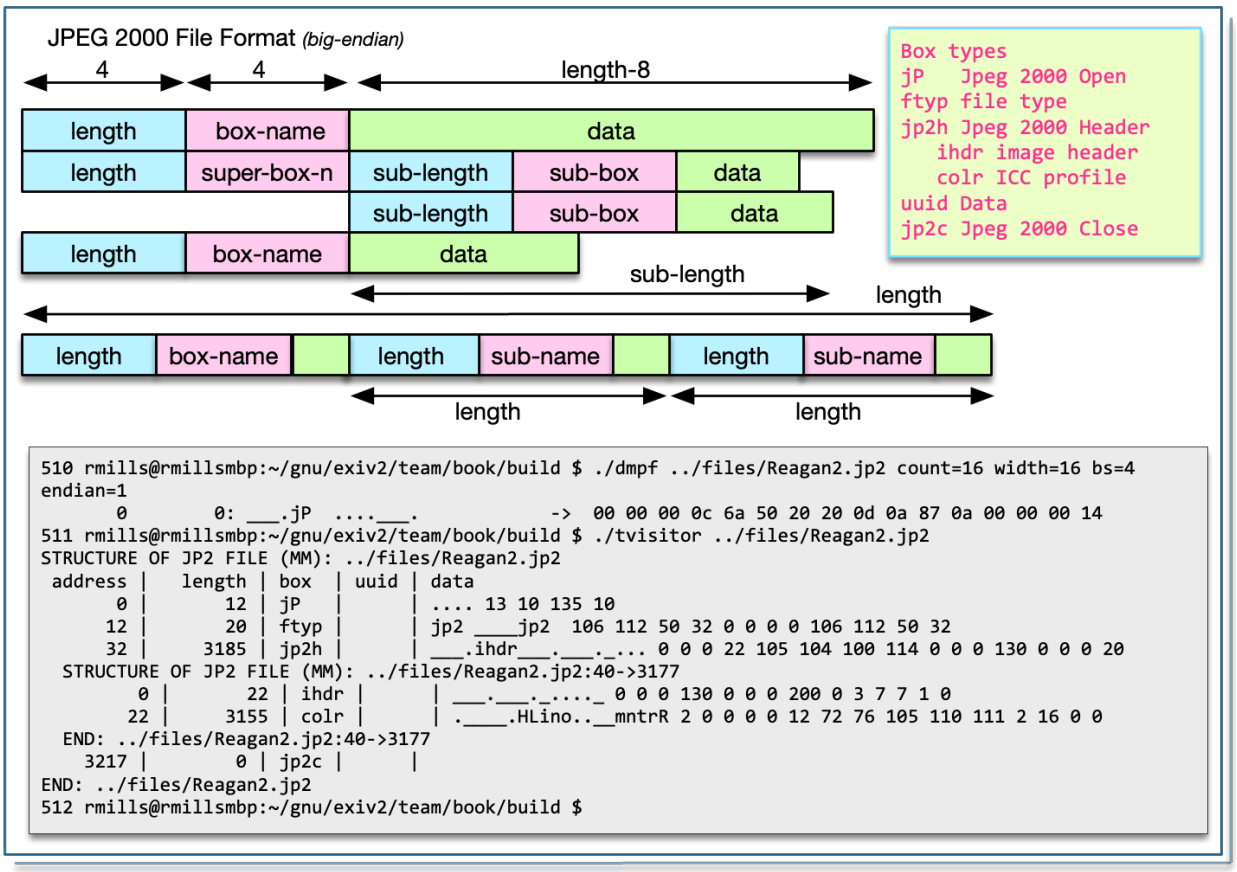
As tvisitor displays the chunks and decompressed data, no further processing is necessary to see this data. However, cannot display this data apart from the zTXt/Description Chunk described below. To support this in Exiv2 requires a new "Family" of metadata with keys such as: Png.zTXt.Author. Adding a new "Family" is a considerable undertaking. The project to have a "unified" metadata container should be undertaken first.

Exiv2 Comment zTXt/Description Chunk

There's an option `$ exiv2 -c abcdefg foo.jpg` which will set the "Comment" in a JPEG file. You can print the comment with `$ exiv2 -pc foo` . A "Comment" in a JPEG is a top level COM segment in the JPEG. Somebody decided to use those commands on a PNG to update an iTXt chunk with the signature "Description".

TOC

JP2 JPEG 2000



JP2 is always big-endian encoded. The documentation is available here:
https://www.iso.org/standard/78321.html

The JPEG 2000 file is an ISOBMFF Container. It consists of a linked lists of “boxes” which have a uint32_t length, char[4] box-type and (length-8) bytes of data. A box may be a “super-box” which is a container for other boxes. A “super-box” can have binary data before the box-chain. Reading the file is very easy, however you need the specification to decode the contents of a box.

I believe the “box” idea in ISOBMFF is intended to address the issue I discussed about TIFF files. In order to rewrite an image, it is necessary for the data to be self contained and relocatable. Every “box” should be self contained with no offsets outside the box. My study of JP2 is restricted to finding the Exiv2, ICC, IPTC and XMP data. For sure these are self-contained blocks of binary data. The metadata boxes are of type uuid and begin with a 128bit/16 byte UUID to identify the data.

In a JP2 the first box, must be box-type of “jP__” and have a length of 12. The chain is terminated with a box-type of “jpcl”. Usually the terminal block with bring you to the end-of-file, however this should not be assumed as there can be garbage following the box chain. The box chain of a super-box is normally terminated by reaching the end of its data.

Validating a JP2 file is straight forward:

```C++
bool Jp2Image::valid()
{
    if ( !valid_ ) {
        start_ = 0;
        IoSave     restore (io(),start_);
        uint32_t   length = io().getLong(endian_);
        uint32_t   box    ;
        io().read(&box,4);
        valid_ = length == 12 && boxName(box) == kJp2Box_jP;
    }
    return valid_ ;
}
```

The accept function is also straight forward:

```C++
void Jp2Image::accept(class Visitor& v)
{
    if ( valid() ) {
        v.visitBegin(*this);
        IoSave restore(io(),start_);
        uint64_t address = start_ ;
        while (  address < io().size() ) {
            io().seek(address );
            uint32_t  length  = io().getLong(endian_);
            uint32_t  box     ;
            io().read(&box,4);
            v.visitBox(io(),*this,address,box,length); // tell the visitor
            // recursion if superbox
            if ( superBox(box) ) {
                uint64_t  subA = io().tell() ;
                Jp2Image jp2(io(),subA,length-8);
                jp2.valid_ = true ;
                jp2.accept(v);
            }
            address = boxName(box) == kJp2Box_jp2c ? io().size() : address + length ;
        }
        v.visitEnd(*this);
    }
}
```

There is a little complication when you create the recursive Jp2Image. We do not wish to validate this because it never starts with box-type of “jP__”. We know the file is valid, so we set the valid_ flag before the recursion.

The function ReportVisitor::visitBox() is also straight forward:

```C++
void ReportVisitor::visitBox(Io& io,Image& image,uint64_t address
                             ,uint32_t box,uint32_t length)
{
    IoSave save(io,address+8);
    length -= 8              ;
    DataBuf   data(length);
    io.read(data);

    std::string name = image.boxName (box);
    std::string uuid = image.uuidName(data);

    if ( option() & (kpsBasic | kpsRecursive) ) {
        out() << indent() << stringFormat("%8d |  %7d | %#10x %4s | %s | ",address,lengt
        if ( length > 40 ) length = 40;
        out() << data.toString(kttUndefined,length,image.endian()) << std::endl;
    }
    if ( option() & kpsRecursive && uuid == "exif" ) {
        Io        tiff(io,address+8+16,data.size_-16); // uuid is 16 bytes (128 bits)
        TiffImage(tiff).accept(*this);
    }
    if ( option() & kpsXMP && uuid == "xmp " ) {
        out() << data.pData_+17 ;
    }
}
```

Although the JP2 file is big endian, the embedded Exif metadata may be little-endian encoded. That's the case with test file Reagan.jp2.

```Bash
.../book/build $ ./tvisitor -pR ../test/data/Reagan.jp2
STRUCTURE OF JP2 FILE (MM): ../test/data/Reagan.jp2
 address |   length | box             | uuid | data
       0 |        4 | 0x2020506a jP   |      | ....
```

```
      12 |       12 | 0x70797466 ftyp |      | jp2 ____jp2
      32 |       37 | 0x6832706a jp2h |      | ___.ihdr___.___._...._____.colr._____.
  STRUCTURE OF JP2 FILE (MM): ../test/data/Reagan.jp2:40->37
   address |   length | box             | uuid | data
         0 |       14 | 0x72646869 ihdr |      | ___.___._...__
        22 |        7 | 0x726c6f63 colr |      | ._____.
  END: ../test/data/Reagan.jp2:40->37
      77 |     1334 | 0x64697575 uuid | exif | JpgTiffExif->JP2II*_.___._..._..__.___..
  STRUCTURE OF TIFF FILE (II): ../test/data/Reagan.jp2:101->1318
   address |    tag                              |      type |    count |    offset | va
        10 | 0x010e Exif.Image.ImageDescription  |     ASCII |      403 |       170 | 04
        22 | 0x010f Exif.Image.Make              |     ASCII |       18 |       574 | NI
...
       142 | 0x8769 Exif.Image.ExifTag           |      LONG |        1 |           | 21
    STRUCTURE OF TIFF FILE (II): ../test/data/Reagan.jp2:101->1318
     address |    tag                                |      type |    count |    offset |
         714 | 0x829a Exif.Photo.ExposureTime        |  RATIONAL |        1 |      1162 |
...
        1122 | 0xa40a Exif.Photo.Sharpness           |     SHORT |        1 |           |
    END: ../test/data/Reagan.jp2:101->1318
    STRUCTURE OF TIFF FILE (II): ../test/data/Reagan.jp2:101->1318
     address |    tag                                |      type |    count |    offset |
        1302 | 000000 Exif.GPSInfo.GPSVersionID      |     UBYTE |        4 |           |
    END: ../test/data/Reagan.jp2:101->1318
  END: ../test/data/Reagan.jp2:101->1318
    1419 |      934 | 0x64697575 uuid | iptc | 3.....G#.......8..__._....._.040621-N-653
    2361 |     5582 | 0x64697575 uuid | xmp  | .z....B..q......<?xpacket begin="..." id
    7951 |    32650 | 0x6332706a jp2c |      | .O.Q_/_____.___.___________.___.________
END: ../test/data/Reagan.jp2
.../book/build $
```

Box Specifications

ICC Profiles in JP2

These are stored in the 'colr' box which is a sub-box of 'jp2h'. I have found the specification very unsatisfactory. w15177_15444 discusses ColourInformationBox extends Box('colr'). I haven't found the definition of 'colr'. I enquired on the ExifTool Forum and Phil offered advice which has been implemented in jp2image.cpp. There are two ways to encode the profile. You can use a `uuid` box with the uuid of "\x01\x00\x00\x00\x00\x10\x00\x00\x05\x1c". The box payload is the ICC profile. Or you can use the 'colr' box which has 3 padding bytes "\02\0\0" followed by the ICC profile. So the length of the box will be 8 (the box) +3 (padding) +iccProfile.size()

I found an older version of the spec in which 'colr' is documented on p161.
http://hosting.astro.cornell.edu/~carcich/LRO/jp2/ISO_JPEG200_Standard/INCITS+ISO+IEC+15444-1-2000.pdf

```Bash
.../book/build $ ./tvisitor ~/gnu/github/exiv2/0.27-maintenance/test/data/Reagan2.
STRUCTURE OF JP2 FILE (MM): /Users/rmills/gnu/github/exiv2/0.27-maintenance/test/data/Re
 address |   length | box             | uuid | data
       0 |        4 | 0x2020506a jP   |      | ....
      12 |       12 | 0x70797466 ftyp |      | jp2 ____jp2
      32 |     3177 | 0x6832706a jp2h |      | ___.ihdr___.___._....___.Scolr._____.HLin
  STRUCTURE OF JP2 FILE (MM): /Users/rmills/gnu/github/exiv2/0.27-maintenance/test/data/
   address |   length | box             | uuid | data
         0 |       14 | 0x72646869 ihdr |      | ___.___._....._
        22 |     3147 | 0x726c6f63 colr |      | ._____.HLino..__mntrRGB XYZ ..._._._._1__
  END: /Users/rmills/gnu/github/exiv2/0.27-maintenance/test/data/Reagan2.jp2:40->3177
    3217 |       -8 | 0x6332706a jp2c |      | .O.Q_/_____.___._____________.___._________
END: /Users/rmills/gnu/github/exiv2/0.27-maintenance/test/data/Reagan2.jp2
.../book/build $
```

As you can see, the 'colr' box is stored at 40+22 bytes into the file and has a length of 3147. The first four bytes of an ICC profile is the length of the file which in this case is 3144 bytes. The next 4 bytes of the profile are the maker and in this case is Linotype.

```Bash
.../book/build $ dmpf skip=$((40+22)) count=19 endian=1  ~/gnu/github/exiv2/0.27-m
   0x3e   62: __.Scolr._____.HLino  ->  00 00 0c 53 63 6f 6c 72 02 00 00 00 00 0c 48 4c 6
                                        <--3147--->  c  o  l  r <-pad--> <--3144--->  L
.../book/build $
```

TOC

ISOBMFF, CR3, HEIF, AVIF

I obtained the standard here: https://mpeg.chiariglione.org/standards/mpeg-4/iso-base-media-file-format/text-isoiec-14496-12-5th-edition

There has been a lot of discussion in Team Exiv2 concerning the legality of reading this file. I don't believe it's illegal to read metadata from a container. I believe it's illegal to decode proprietary encoded data stored in the image. However the metadata is not protected in anyway. So, I'll implement this in tvisitor.cpp. Team Exiv2 may agree to include this in Exiv2 v0.28. If I ever work on Exiv2 v0.27.4, I will implement ISOBMFF by extending the existing JP2 code.

The most obvious difference between JP2000 and ISOBMFF is the first box. For JP2, this is of type **jP** *(jPspacespace)* followed by **ftyp**. ISOBMFF files begin with an **ftyp** box. The syntax of the **ftyp** box is:

```
aligned(8) class FileTypeBox  extends Box('ftyp') {
  unsigned int(32) major_brand;
  unsigned int(32) minor_version;
  unsigned int(32) compatible_brands[]; // to end of the box
}
```

So there are two uint32_t values which are the brand and minor_version. Then zero or more uint32_t values for compatible brands.

Box Names

A box name is a 4 byte big-endian byte stream and stored in a uint32_t. It is not nul-terminated. So the box type **jP** *(jPspacespace)* is 0x2020506a, and **ftyp** is 0x70794666.

UUID Box uuid

This is mechanism to store binary data in any format. The ISOBMFF Specification states: *Type Fields not defined here are reserved. Private extensions shall be achieved through the 'uuid' type.* The uuid box has a 128 bit (16 byte) UUID to identify the data, followed by the data. This is similar to the "signature" in JPEG segment or PNG chunk.

ISOBMFF Explorer

I've found the open-source product ISOBMFF Explorer very useful in learning about this file format. https://imazing.com/isobmff/download. The code is available from: (https://github.com/DigiDNA/ISOBMFF)[https://github.com/DigiDNA/ISOBMFF].

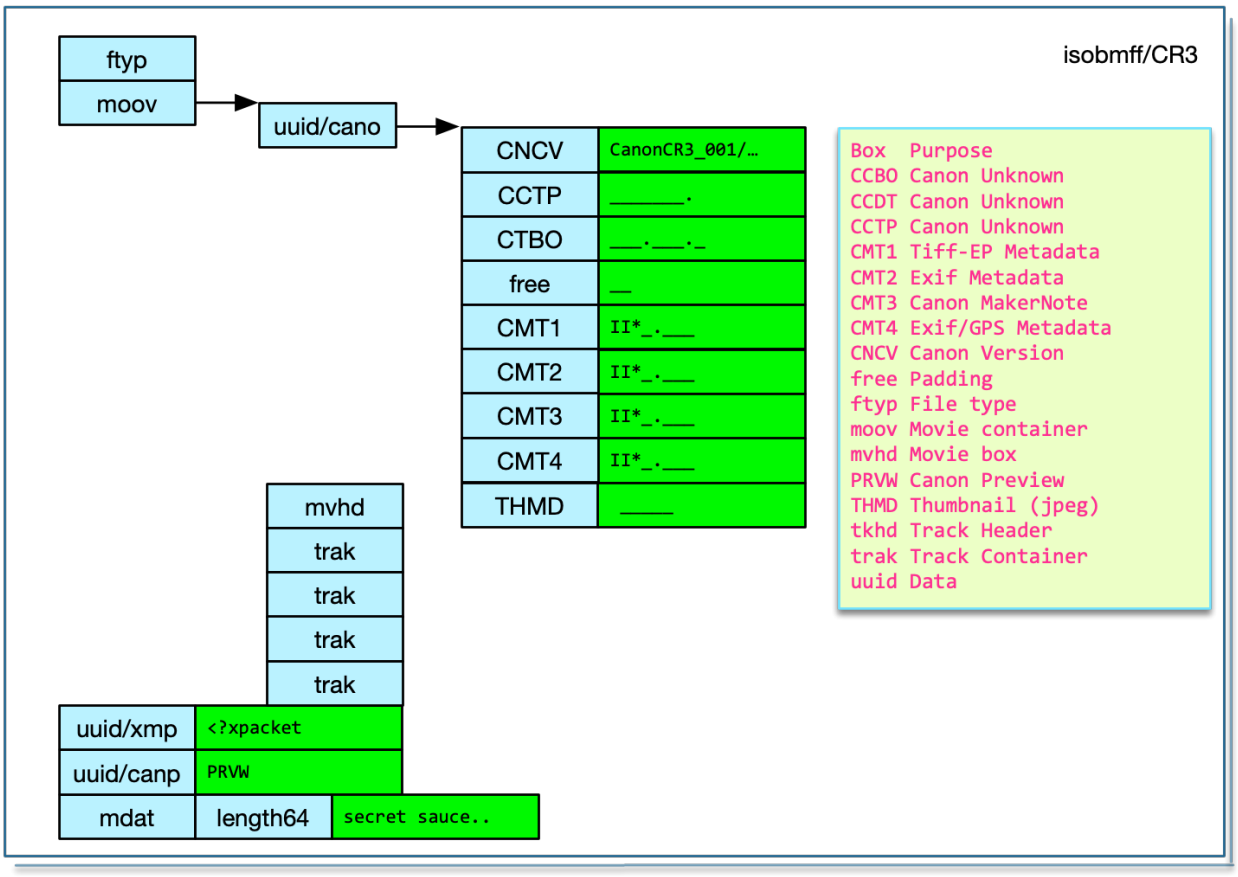
I built it as follows. You should use the git –recursive option to ensure that Submodules are also cloned.

```Bash
$ git clone --recursive https://github.com/DigiDNA/ISOBMFF --depth 1
$ open ISOBMFF.xcodeproj/
```

Very nice program with very nice code. In addition to the GUI/Explorer, there is command-line utility ISOBMFF-Dump provided. I also built it with Visual Studio 2019. I believe the GUI is only provided on the Mac. The command-line program is supported on Mac, Windows and Linux.

It did not build *out of the box* for me on Ubuntu18.04. [https://github.com/DigiDNA/ISOBMFF/issues/12] (https://github.com/DigiDNA/ISOBMFF/issues/12}

Canon CR3 Format



This is a dump from a CR3:

```
$ tvisitor ~/cr3.cr3
STRUCTURE OF CR3 FILE (MM): /Users/rmills/cr3.cr3
 address |   length | box             | uuid | data
       0 |       16 | 0x70797466 ftyp |      | crx ___.crx isom
      24 |    22784 | 0x766f6f6d moov |      | __PXuuid............F+jH___&CNCVCanonCR3
  STRUCTURE OF JP2 FILE (MM): /Users/rmills/cr3.cr3:32->22784
         0 |    20560 | 0x64697575 uuid | can1 | ___&CNCVCanonCR3_001/00.09.00/00.00.00_
    STRUCTURE OF JP2 FILE (MM): /Users/rmills/cr3.cr3:32->22784:24->20552
           0 |       30 | 0x56434e43 CNCV |      | CanonCR3_001/00.09.00/00.00.00
          38 |       84 | 0x50544343 CCTP |      | _______.___.___.CCDT_______._______.
         130 |       84 | 0x4f425443 CTBO |      | ___.___._____Y _____._.___._____.Y8_
         222 |        2 | 0x65657266 free |      | __
         232 |      384 | 0x31544d43 CMT1 |      | II*_.___.__..._.___p.__..._.___..__...
         624 |     1056 | 0x32544d43 CMT2 |      | II*_.___._..._.___..__..._.___..__"...
        1688 |     5168 | 0x33544d43 CMT3 |      | II*_.___/_._._1___B.__._._.___..__._.
        6864 |     1808 | 0x34544d43 CMT4 |      | II*_.___.___._.___..________________
        8680 |    11856 | 0x424d4854 THMB |      | _____._x__.=._.__...._._............
       20544 |      100 | 0x6468766d mvhd |      | ____..4...4.___.___._.__.___________
    END: /Users/rmills/cr3.cr3:32->22784:24->20552
     20568 |      100 | 0x6468766d mvhd |      | ____..4...4.___.___._.__.___________..
     20676 |      476 | 0x6b617274 trak |      | ___\tkhd___...4...4.___._______._______
     21160 |      576 | 0x6b617274 trak |      | ___\tkhd___...4...4.___._______._______
     21744 |      592 | 0x6b617274 trak |      | ___\tkhd___...4...4.___._______._______
     22344 |      432 | 0x6b617274 trak |      | ___\tkhd___...4...4.___._______._______
  END: /Users/rmills/cr3.cr3:32->22784
   22816 |    65552 | 0x64697575 uuid |  xmp | <?xpacket begin='...' id='W5M0MpCehiHzre
   88376 |   264921 | 0x64697575 uuid | can2 | _______._...PRVW_____..T.8_._......_._.
  353305 |       -7 | 0x7461646d mdat |      | _____.j....._._.....................
END: /Users/rmills/cr3.cr3
```

The CR3 format has been well documented by Laurent Clévy here: https://github.com/lclevy/canon_cr3.git

The XMP is clearly marked in a uuid packet. The Exif metadata is stored as 4 embedded TIFF files in the Canon uuid packet. The four files are the 'tiffTags', 'exifTags', 'canonTags' and 'gpsTags'. In the test files, the gpsTags are about 1800 bytes of mostly zeros!

Laurent hasn't identified IPTC and ICC data. There is a discussion about concerning ICC in JP2000 files and I believe that's what is used by CR3. I have not discovered anything about IPTC in CR3 files.

The THMB record is a JPEG and written at an offset of 24 bytes into the record.

```
$ (dd bs=1 skip=32 count=22784 if=~/cr3.cr3 | dd bs=1 skip=24 count=20552 | dd bs=
STRUCTURE OF JPEG FILE: 1596571254.exiv2_temp
 address | marker       |  length | data
       0 | 0xffd8 SOI
       2 | 0xffdb DQT   |     132
     136 | 0xffc0 SOF0  |      17
     155 | 0xffc4 DHT   |     418
     575 | 0xffda SOS
```

In this case, the thumbnail is 160x120 pixels. 11837 is the filesize.

```
$ (dd bs=1 skip=32 count=22784 if=~/cr3.cr3 | dd bs=1 skip=24 count=20552 | dd bs=1 skip
       0        0: __.XTHMB_____._x__.=_.__          ->      0 11864 21576 19778     0
$ ls -l foo.jpg
-rw-r--r--@ 1 rmills  staff  11832  4 Aug 20:58 foo.jpg
```

Laurent has documented this as: **THMB** *(Thumbnail)* from **uuid** = 85c0b687-820f-11e0-8111-f4ce462b6a48

Offset	type	size	content
0	long	1	size of this tag
4	char	4	“THMB”
8	byte	1	likely version, value=0 or 1
9	bytes	3	likely flags, value = 0

for version 0:

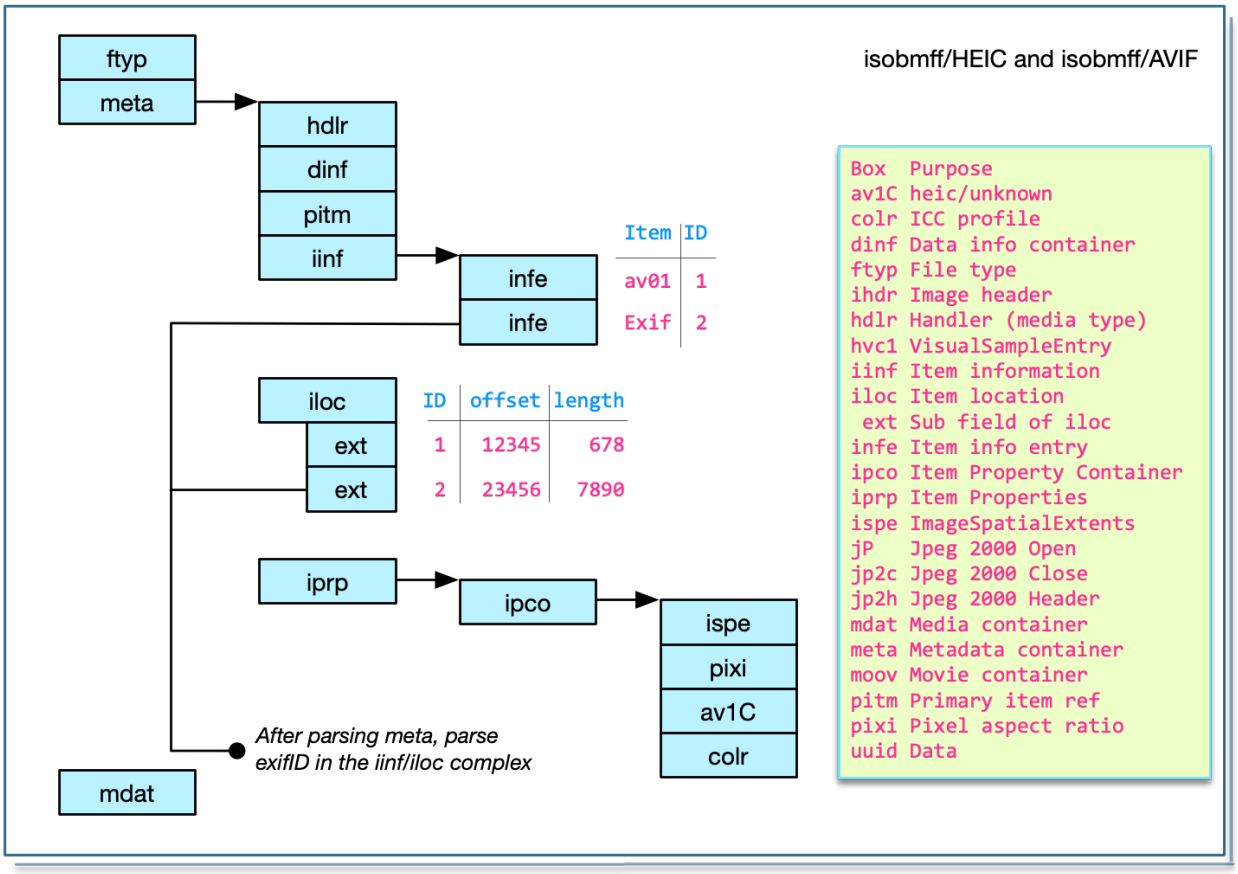
Offset	type	size	content
12/0xc	short	1	width (160)
14/0xe	short	1	height (120)
16/0x10	long	1	jpeg image size (jpeg_size)
20/0x14	short	1	unknown, value = 1
22/0x16	short	1	unknown, value = 0
24/0x18	byte[]	stored at offset 16	jpeg_data = ffd8ffdb…ffd9
24+jpeg_size	byte[]	?	padding to next 4 bytes?
	long	1	?

for version 1:

Offset	type	size	content
12/0xc	short	1	width (160)
14/0xe	short	1	height (120)
16/0x10	long	1	jpeg image size (jpeg_size)

HEIC and AVIF

To understand how to parse HEIC and AVIF, we have to discuss the specification of more boxes. The file is organized into a heirarchy of box as shown in this drawing.



Full Box

The Full Box is specified as follows:

```
aligned(8) class FullBox(unsigned int(32) boxtype, unsigned int(8) v, bit(24) f)
 extends Box(boxtype) {
 unsigned int(8) version = v;
 bit(24) flags = f;
}
```

A "Full Box" has a 4 byte header which is version (1 byte) followed by flags (3 bytes).

Handler Box

This is specified as follows:

```
aligned(8) class HandlerBox extends FullBox('hdlr', version = 0, 0) {
 unsigned int(32) pre_defined = 0;
 unsigned int(32) handler_type;
 const unsigned int(32)[3] reserved = 0;
 string name;
}
```

Media Box mdat

This is specified as follows:

```
aligned(8) class MediaDataBox extends Box('mdat') {
   bit(8) data[];
}
```

This is pure binary data. From a metadata perspective, this the end of the file.

Meta Box meta

This is specified as follows:

```
aligned(8) class MetaBox (handler_type)
   extends FullBox('meta', version = 0, 0) {
   HandlerBox(handler_type) theHandler;
   PrimaryItemBox
   DataInformationBox
   ItemLocationBox
   ItemProtectionBox
   ItemInfoBox
   IPMPControlBox
   ItemReferenceBox
   ItemDataBox
   Box other_boxes[];
}
```

Item Information Box iinf

The Item information box provides extra information about selected items, including symbolic (File) names.

This is specified as follows:

```
aligned(8) class ItemInfoBox
  extends FullBox('iinf', version, 0) {

  if (version == 0) {
    unsigned int(16) entry_count;
  } else {
    unsigned int(32) entry_count;
  }
  ItemInfoEntry[ entry_count ]
}
```

Item Location Box iloc

The item location box provides a directory of resources in this or other Files, by locating their container, their offset within that container, and their length. Placing this in binary format enables common handling of this data, even by systems which do not understand the particular metadata system used.

This is specified as follows:

```
aligned(8) class ItemLocationBox extends FullBox('iloc', version, 0) {
  unsigned int(4)
  unsigned int(4)
  unsigned int(4)
  if ((version == 1) || (version == 2)) {
     offset_size;
     length_size;
     base_offset_size;
     unsigned int(4) index_size;
  } else {
    unsigned int(4) reserved;
  }
  if (version < 2) {
    unsigned int(16) item_count;
  } else if (version == 2) {
    unsigned int(32) item_count;
  }
  for (i=0; i<item_count; i++) {
    if (version < 2) {
      unsigned int(16) item_ID;
    } else if (version == 2) {
      unsigned int(32) item_ID;
    }
    if ((version == 1) || (version == 2)) {
      unsigned int(12) reserved = 0;
      unsigned int(4) construction_method;
    }
    unsigned int(16) data_reference_index;
    unsigned int(base_offset_size*8) base_offset;
    unsigned int(16) extent_count;
    for (j=0; j<extent_count; j++) {
      if (((version == 1) || (version == 2)) && (index_size > 0)) {
        unsigned int(index_size*8) extent_index;
      }
      unsigned int(offset_size*8) extent_offset;
      unsigned int(length_size*8) extent_length;
    } // for j
  } // for i
}
```

Test HEIC File

I obtained HEIC test files from:
https://github.com/thorsted/digicam_corpus/tree/master/Apple/iPhone%20XR

I dumped IMG_3578.HEIC with dmpf and disassembled it by hand:

```
       0        0: ___ ftypheic____mif1  ->  00 00 00 20 66 74 79 70 68 65 69 63 00 00 0
                                            <  length >  f  t  y  p <   brand > < minor
    0x14       20: miafMiHBheic__.4meta  ->  6d 69 61 66 4d 69 48 42 68 65 69 63 00 00 0
                                             m  i  a  f  M  i  H  B  h  e  i  c <  leng
    0x28       40: _______.hdlr________  ->  00 00 00 00 00 00 00 22 68 64 6c 72 00 00 0
                                            < version?> < length  >  h  d  l  r < versi
    0x3c       60: pict________________  ->  70 69 63 74 00 00 00 00 00 00 00 00 00 00 0
                                             p  i  c  t
    0x50       80: _$dinf___.dref______  ->  00 24 64 69 6e 66 00 00 00 1c 64 72 65 66 0
                                            ngth>  d  i  n  f <  length > d  r  e  f
    0x64      100: _.___.url ___.___.pi  ->  00 01 00 00 00 0c 75 72 6c 20 00 00 00 01 0
                                                  <  length >  u r  l  sp            <
    0x78      120: tm_____1__.=iinf____  ->  74 6d 00 00 00 00 00 31 00 00 04 3d 69 69 0
                                             t  m                  <  length >  i  i
    0x8c      140: _3___.infe.__._.__hv  ->  00 33 00 00 00 15 69 6e 66 65 02 00 00 01 0
                                            < E#> <  length > i  n  f  e  < V > <flag><
    0xa0      160: c1____.infe.__._.__h  ->  63 31 00 00 00 00 15 69 6e 66 65 02 00 00 0
                                             c  1    <  length >  i  n  f  e
    0xb4      180: vc1____.infe.__._.__  ->  76 63 31 00 00 00 00 15 69 6e 66 65 02 00 0
...
   0x44c     1100: .__hvc1____.infe.__.  ->  2e 00 00 68 76 63 31 00 00 00 00 15 69 6e 0
   0x460     1120: _/__hvc1____.infe.__  ->  00 2f 00 00 68 76 63 31 00 00 00 00 15 69 0
   0x49c     1180: ____2__hvc1____.infe  ->  00 00 00 00 32 00 00 68 76 63 31 00 00 00 0
                                                                                  <  leng
   0x4b0     1200: .__._3__Exif____.ire  ->  02 00 00 01 00 33 00 00 45 78 69 66 00 00 0
                                            < V > <flag>< ID> <pro>  E  x  i  fnul <  1
   0x4c4     1220: f_______ldimg_1_0_._  ->  66 00 00 00 00 00 00 00 6c 64 69 6d 67 00 3
                                             f
   0x4d8     1240: ._._._._._._._._._._  ->  02 00 03 00 04 00 05 00 06 00 07 00 08 00 0
...
   0xa14     2580: __.@iloc.___D__3_.__  ->  00 00 03 40 69 6c 6f 63 01 00 00 00 44 00 0
                                            <  length >  i  l  o  c < FullBox > <o-l> <
   0xa28     2600: ___.__*.__..._._____.  ->  00 00 00 01 00 00 2a b3 00 00 2e a7 00 02 0
                                            <DRI> <OFF> <EXC> 10931 < ??> 11943 < ID> <
   0xa3c     2620: __YZ__`._._____.__.K  ->  00 00 59 5a 00 00 60 f1 00 03 00 00 00 00 0
                                            <EXC> 22874  ?  ? 24817 < ID> < CM> <DRI> <
   0xa50     2640: __N._._____._...__K.  ->  00 00 4e 8e 00 04 00 00 00 00 01 00 01 0
   0xa64     2660: _._____._.T.__M._.__  ->  00 05 00 00 00 00 01 00 01 54 d2 00 00 4
   0xa78     2680: ___._...__N._._____.  ->  00 00 00 01 00 01 a1 d4 00 00 4e ff 00 07 0
   0xa8c     2700: _..._F._._____._.7.  ->  00 01 f0 d3 00 00 46 cc 00 08 00 00 00 00 0

   EXC = extent_count   CM = construction_method #E = Number of Entries  DRI = data_reve
   OFF = base_offset   o-l = offset-length       ID = Indentifier
```

The code in ISOBMFF/iloc.cpp is (effectively):

```C++
void ILOC::ReadData( Parser & parser, BinaryStream & stream )
{
    FullBox::ReadData( parser, stream );

    uint8_t u8 = stream.ReadUInt8();
    this->SetOffsetSize( u8 >> 4 );
    this->SetLengthSize( u8 & 0xF );

    u8 = stream.ReadUInt8();
    this->SetBaseOffsetSize( u8 >> 4 );
    this->SetIndexSize( u8 & 0xF );
    uint32_t count = this->GetVersion() < 2 ) ?stream.ReadBigEndianUInt16() : stream.Rea

    this->impl->_items.clear();
    for( uint32_t i = 0; i < count; i++ )
    {
        this->AddItem( std::make_shared< Item >( stream, *( this ) ) );
    }
}
```

When processed by tvisitor, we see:

```Bash
$ tvisitor ~/Downloads/IMG_3578.HEIC
STRUCTURE OF JP2 (heic) FILE (MM): /Users/rmills/Downloads/IMG_3578.HEIC
 address |   length |  box | uuid | data
       0 |       24 | ftyp |      | heic____mif1miafMiHB
      32 |     3372 | meta |      | _______.hdlr________   'meta' box
  STRUCTURE OF JP2 FILE (MM): /Users/rmills/Downloads/IM
         0 |       26 | hdlr |      | ________pict______
        34 |       28 | dinf |      | ___.dref_______.__   'meta/dinf' box
    STRUCTURE OF JP2 FILE (MM): /Users/rmills/Downloads/
           0 |       20 | dref |      | _______.___.url
    END: /Users/rmills/Downloads/IMG_3578.HEIC:44->3368:
        70 |        6 | pitm |      | _____1 0 0 0 0 0 4
        84 |     1077 | iinf |      | _____3___.infe.__.   'meta/dinf/iinf' box
    STRUCTURE OF JP2 FILE (MM): /Users/rmills/Downloads/
           0 |       13 | infe |      | .__._.__hvc1_      'meta/dinf/iinf/infe' boxes
...
        1050 |       13 | infe |      | .__._3__Exif_       ID==51 "Exif\0"
    END: /Users/rmills/Downloads/IMG_3578.HEIC:44->3368:
      1169 |      140 | iref |      | _______ldimg_1_0_.
      1317 |     1195 | iprp |      | __.lipco__.0colrpr
    STRUCTURE OF JP2 FILE (MM): /Users/rmills/Downloads/
           0 |      868 | ipco |      | __.0colrprof__.$
      STRUCTURE OF JP2 FILE (MM): /Users/rmills/Download
             0 |      552 | colr |      | prof__.$appl._
           560 |      104 | hvcC |      | ..p___._____Z.
           672 |       12 | ispe |      | ______.___._ 0
           692 |       12 | ispe |      | ______..__.. 0
           712 |        1 | irot |      | _ 0
           721 |        8 | pixi |      | ____.... 0 0 0
           737 |      103 | hvcC |      | ..p___._____<.
           848 |       12 | ispe |      | ______.@___. 0
      END: /Users/rmills/Downloads/IMG_3578.HEIC:44->336
         876 |      311 | ipma |      | _______2_.....-.
    END: /Users/rmills/Downloads/IMG_3578.HEIC:44->3368:
      2520 |        8 | idat |      | __..... 0 0 5 7 1
      2536 |      824 | iloc |      | .___D__3_._____.__   'meta/dinf/iloc' box
      2544 |       16 |  ext |    1 |  10931, 11943         iloc is a binary array
...
      3344 |       16 |  ext |   51 |   8907,  2024         ID==51 offset/length
  END: /Users/rmills/Downloads/IMG_3578.HEIC:44->3368
    3412 |       -7 | mdat |      | _____..__.c(........  'mdat' box (EOF)
  STRUCTURE OF TIFF FILE (MM): /Users/rmills/Downloads/IM Dump the embedded Exif/TIFF
   address |    tag                                |      t
        10 | 0x010f Exif.Image.Make                |     AS
        22 | 0x0110 Exif.Image.Model               |     AS
...
    END: /Users/rmills/Downloads/IMG_3578.HEIC:8917->2014
  END: /Users/rmills/Downloads/IMG_3578.HEIC:8917->2014
END: /Users/rmills/Downloads/IMG_3578.HEIC
```

Reporting Boxes as Metadata

The tvisitor.cpp code is mostly a structural parser. It locates Exif metadata within the meta box and, if the user has selected the Recursive option (-pR), will report the Exif metadata. However, the basic report can treat

boxes as metadata and this has been done for the ispe box which is specified as follows:

```
6.5.3.2 Syntax
aligned(8) class ImageSpatialExtentsProperty
extends ItemFullProperty('ispe', version = 0, flags = 0) {
unsigned int(32) image_width;
       unsigned int(32) image_height;
}
```

This is coded into tvisitor.cpp as follows:

```C++
    // ISOBMFF boxes
    boxDict["ispe"] = "ISOBMFF.ispe";
    boxTags["ispe"].push_back(Field("Version"         ,kttUShort , 0, 1));
    boxTags["ispe"].push_back(Field("Flags"           ,kttUShort , 2, 1));
    boxTags["ispe"].push_back(Field("Width"           ,kttLong   , 4, 1));
    boxTags["ispe"].push_back(Field("Height"          ,kttLong   , 8, 1));
```

The processing of this data is achieved in ReportVisitor::visitBox() as follows:

```
    if ( boxDict.find(name) != boxDict.end() ) {
        if ( boxTags.find(name) != boxTags.end() ) {
            for (Field field : boxTags[name] ) {
                std::string n      = chop( boxDict[name] + "." + field.name(),28);
                endian_e    endian = field.endian() == keImage ? image.endian() : field.
                out() << indent() << stringFormat("%-28s ",n.c_str())
                      << chop(data.toString(field.type(),field.count(),endian,field.star
                      << std::endl;
            }
        }
    }
```

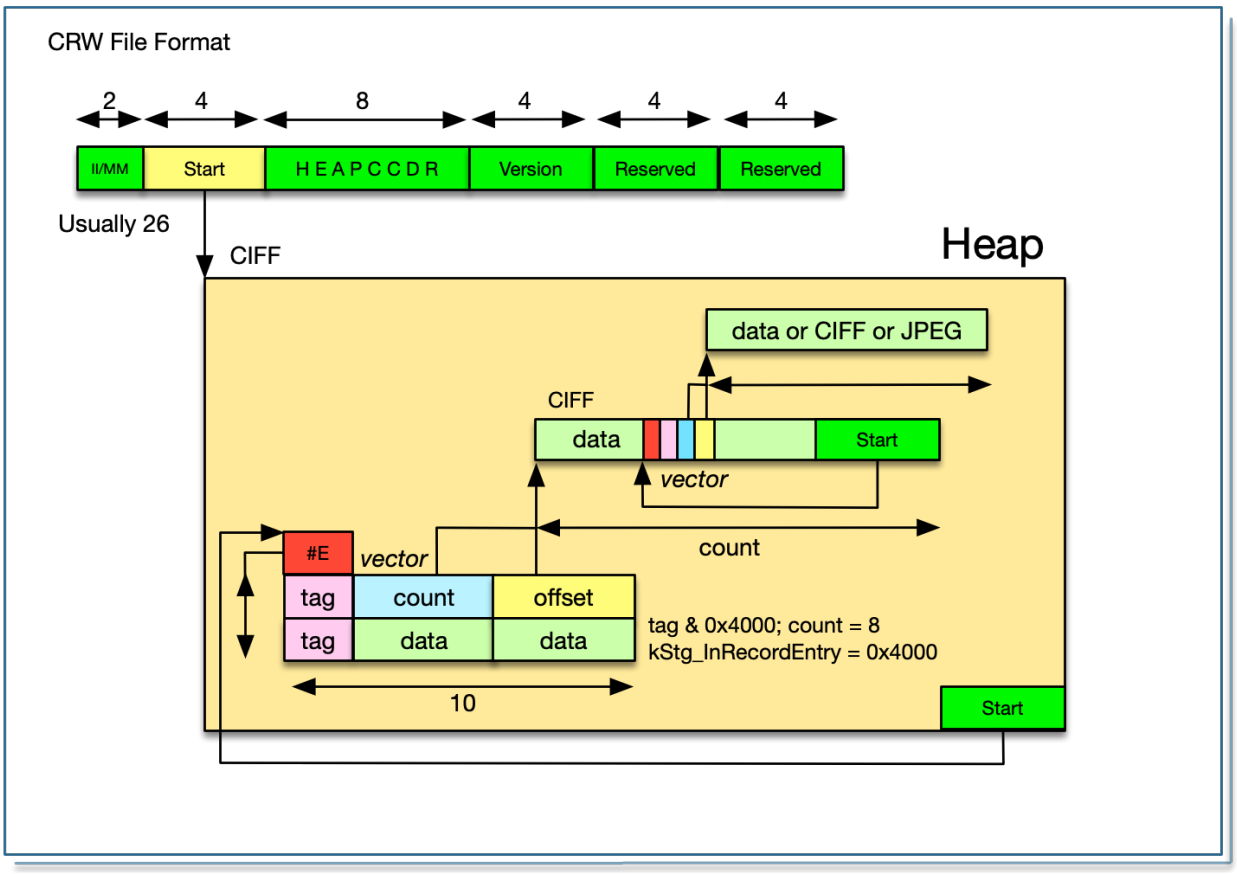
These tags are reported as metadata as follows:

```
           692 |       12 | ispe |      | ______..__.. 0 0 0 0 0 0 15 192 0 0 11 208
      ISOBMFF.ispe.Version         0
      ISOBMFF.ispe.Flags           0
      ISOBMFF.ispe.Width           4032
      ISOBMFF.ispe.Height          3024
```

More information about binary decoding in tvisitor.cpp is discussed in 3.5 ReportVisitor::visitTag()

TOC

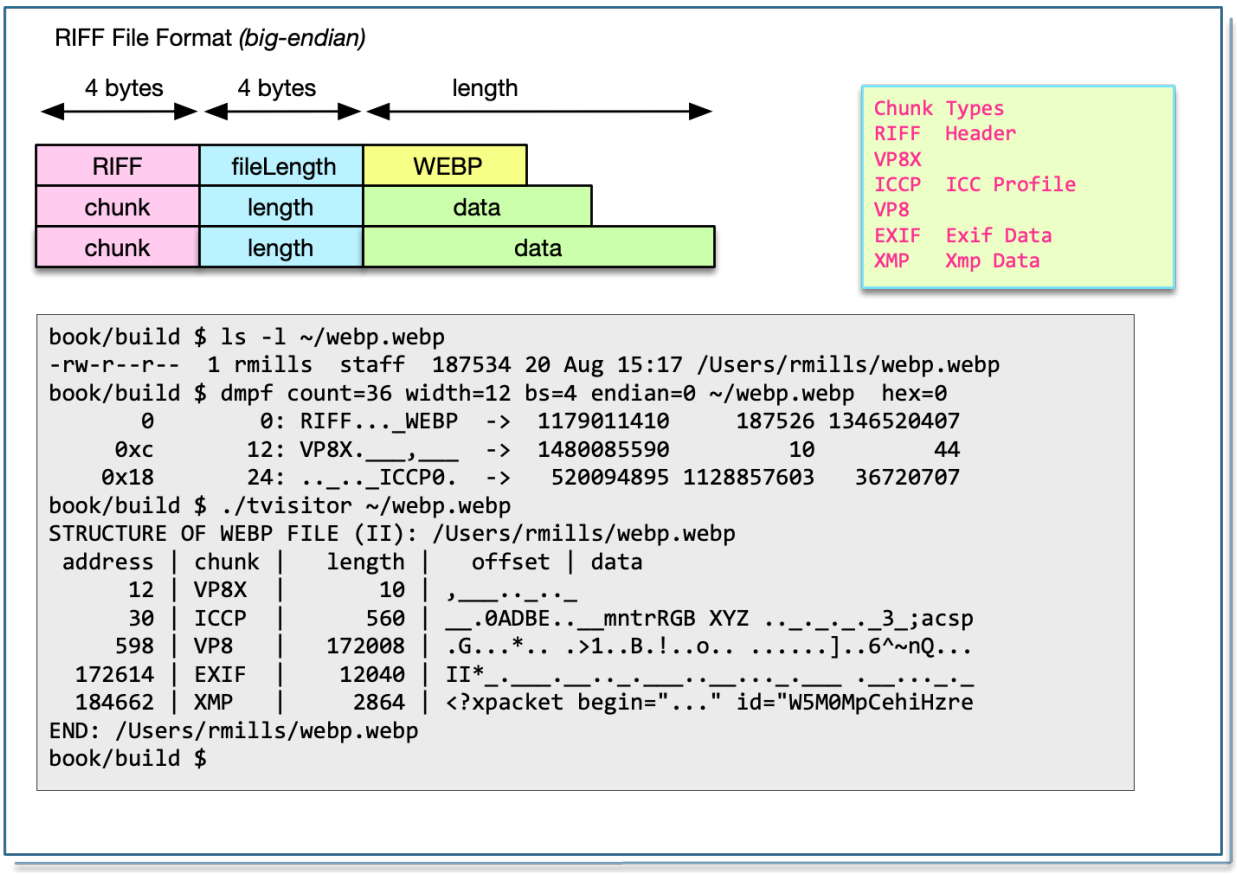
CRW Canon Raw Format



The specification is here: CIFFspecV1R04.pdf

TOC

RIFF Resource Interchange File Format



This file format is used by WEBP, AVI and WAV files. RIFF was introduced in 1991 by Microsoft and IBM. There is a discussion of the format here: https://en.wikipedia.org/wiki/Resource_Interchange_File_Format

Testing for validity is easy:

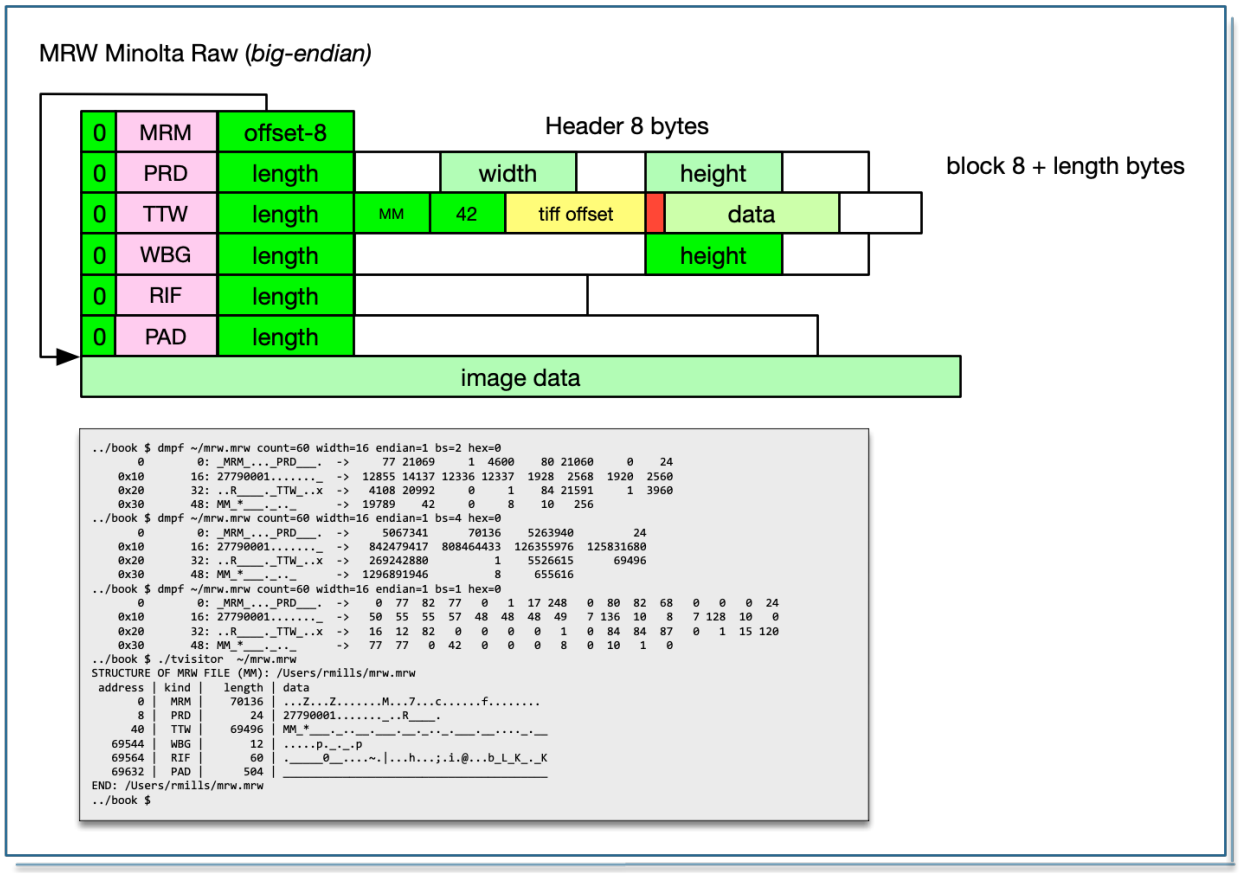
```C++
bool valid()
{
    if ( !valid_ ) {
        IoSave  restore(io(),0);
        DataBuf header(12);
        io().read(header);
        fileLength_   = ::getLong(header,4,endian_);
        valid_        =  header.begins("RIFF") && fileLength_ <= io().size();
        char            signature[5];
        format_       =  header.getChars(8,4,signature);
        header_       = " address | chunk |   length |   offset | data " ;
    }
    return valid_;
}
```

Reporting the data in the file is straight-forward:

```C++
void RiffImage::accept(class Visitor& visitor)
{
    if ( !valid_ ) valid();
    if (  valid_ ) {
        visitor.visitBegin((*this)); // tell the visitor

        IoSave   restore(io(),start_);
        uint64_t address = start_;
        DataBuf  riff(8);
        DataBuf  data(40);  // buffer to pass data to visitRiff()
        while (  address < fileLength_ ) {
            visit(address);
            io().seek(address);
            io().read(riff);

            char        signature[5];
            std::string chunk   = riff.getChars(0,4,signature);
            uint32_t    length  = ::getLong(riff,4,endian_) ;
            uint64_t    pad     = length % 2 ? 1 : 0        ; // pad if length is odd
            uint64_t    next    = io().tell() + length +pad ;
            if ( next > fileLength_ ) Error(kerCorruptedMetadata);

            data.zero();
            io().read(data.pData_,length < data.size_?length:data.size_);
            visitor.visitRiff(address,chunk,length,data);

            if ( chunk == "XMP " || chunk == "ICCP" ) {
                DataBuf    Data(length);
                io().seek(address+8);
                io().read(Data);
                if ( chunk == "XMP "    ) visitor.visitXMP(Data);
                if ( chunk == "ICCP"    ) visitor.visitICC(Data);
            }
            if ( chunk == "EXIF" ) {
                Io tiff(io(),address+8,length);
                visitor.visitExif(tiff);
            }
            address = next ;
        }
        visitor.visitEnd((*this)); // tell the visitor
    }
}
```

TOC

MRW Minolta Raw Format



There is information about this format here: http://www.dalibor.cz/software/minolta-raw-mrw-file-format

```Bash
659 rmills@rmillsmbp:~/gnu/exiv2/team/book $ dmpf ~/mrw.mrw count=60 width=16 endi
       0        0: _MRM_..._PRD___.  ->     77 21069     1  4600    80 21060     0    24
    0x10       16: 27790001.......x  ->  12855 14137 12336 12337  1928  2568  1920  2560
    0x20       32: ..R____._TTW_..x  ->   4108 20992     0     1    84 21591     1  3960
    0x30       48: MM_*___._..._     ->  19789    42     0     8    10   256
660 rmills@rmillsmbp:~/gnu/exiv2/team/book $
```

TOC

ORF Olympus Raw Format

This is a member of the TIFF family of formats.

```
.../book 509 rmills@rmillsmbp:~/gnu/exiv2/team/book $ dmpf count=40 width=20 endia
       0        0: IIRO.___.__..._.___..  ->  4949 4f52    8    0   15  100    4    1
                                               II magic   offset   E#  tag   type   cou
    0x14       20: __...._.___..__...._._  ->     0  101    4    1    0  c0c    0  102
.../book $
```

The following is a typical dump:

```Bash
STRUCTURE OF TIFF FILE (II): /Users/rmills/ORF.ORF
 address |    tag                              |      type |    count |    offset | valu
      10 | 0x0100 Exif.Image.ImageWidth        |      LONG |        1 |           | 4100
      22 | 0x0101 Exif.Image.ImageLength       |      LONG |        1 |           | 3084
      34 | 0x0102 Exif.Image.BitsPerSample     |     SHORT |        1 |           | 16
...
     238 | 0x8769 Exif.Image.ExifTag           |      LONG |        1 |           | 266
  STRUCTURE OF TIFF FILE (II): /Users/rmills/ORF.ORF
   address |    tag                              |      type |    count |    offset | va
       268 | 0x829a Exif.Photo.ExposureTime      |  RATIONAL |        1 |      3332 | 1/
       280 | 0x829d Exif.Photo.FNumber           |  RATIONAL |        1 |      3372 | 10
...
       424 | 0x927c Exif.Photo.MakerNote         | UNDEFINED |  1452144 |      3472 | OL
    STRUCTURE OF  FILE (II): /Users/rmills/ORF.ORF:3472->1452144
          14 | 0x0100 Exif.Olympus.ThumbnailImage | UNDEFINED |     4892 |     11792 |
          26 | 0x0200 Exif.Olympus.SpecialMode     |      LONG |        3 |      4256 |
          50 | 0x2010 Exif.Olympus.Equipment       |       IFD |        1 |           |
      STRUCTURE OF  FILE (II): /Users/rmills/ORF.ORF:3472->1452144
           116 | 000000 Exif.OlympusEQ.Version       | UNDEFINED |        4 |
           128 | 0x0100 Exif.OlympusEQ.CameraType    |     ASCII |        6 |      4304
           140 | 0x0101 Exif.OlympusEQ.SerialNumber  |     ASCII |       32 |      4310
           152 | 0x0102 Exif.OlympusEQ.InternalSer.. |     ASCII |       32 |      4342
      END: /Users/rmills/ORF.ORF:3472->1452144
      STRUCTURE OF  FILE (II): /Users/rmills/ORF.ORF:3472->1452144
           410 | 000000 Exif.OlympusCS.Version       | UNDEFINED |        4 |
           422 | 0x0100 Exif.OlympusCS.PreviewImag.. |      LONG |        1 |
           434 | 0x0101 Exif.OlympusCS.PreviewImag.. |      LONG |        1 |
           446 | 0x0102 Exif.OlympusCS.PreviewImag.. |      LONG |        1 |
           458 | 0x0200 Exif.OlympusCS.ExposureMode  |     SHORT |        1 |
      END: /Users/rmills/ORF.ORF:3472->1452144
      STRUCTURE OF  FILE (II): /Users/rmills/ORF.ORF:3472->1452144
          1076 | 000000 Exif.OlymRawDev.Version      | UNDEFINED |        4 |
          1088 | 0x0100 Exif.OlymRawDev.ExposureBi.. | SRATIONAL |        1 |      5304
          1100 | 0x0101 Exif.OlymRawDev.WhiteBalan.. |     SHORT |        1 |
          1112 | 0x0102 Exif.OlymRawDev.WBFineAdju.. |    SSHORT |        1 |
          1124 | 0x0103 Exif.OlymRawDev.GrayPoint    |     SHORT |        3 |      5316
      END: /Users/rmills/ORF.ORF:3472->1452144
      STRUCTURE OF  FILE (II): /Users/rmills/ORF.ORF:3472->1452144
          1250 | 000000 Exif.OlymImgProc.Version     | UNDEFINED |        4 |
          1262 | 0x0100 Exif.OlymImgProc.WB_RBLevels |     SHORT |        4 |      5354
      END: /Users/rmills/ORF.ORF:3472->1452144
      STRUCTURE OF  FILE (II): /Users/rmills/ORF.ORF:3472->1452144
          3488 | 000000 Exif.OlymFocusInfo.Version   | UNDEFINED |        4 |
          3596 | 0x0209 Exif.OlymFocusInfo.AutoFocus |     SHORT |        1 |
          3680 | 0x0210 Exif.OlymFocusInfo.SceneDe.. |     SHORT |        1 |
          3692 | 0x0211 Exif.OlymFocusInfo.SceneArea |      LONG |        8 |      8040
      END: /Users/rmills/ORF.ORF:3472->1452144
    END: /Users/rmills/ORF.ORF:3472->1452144
       436 | 0x9286 Exif.Photo.UserComment       | UNDEFINED |      125 |      3164 | __
       448 | 0xa000 Exif.Photo.FlashpixVersion   | UNDEFINED |        4 |           | 01
       460 | 0xa001 Exif.Photo.ColorSpace        |     SHORT |        1 |           | 1
...
  END: /Users/rmills/ORF.ORF
END: /Users/rmills/ORF.ORF
```

The MakerNote contains almost all the data in the file:

```
424 | 0x927c Exif.Photo.MakerNote          | UNDEFINED |  1452144 |      3472 | OLYMPUS_
```

It consists of and single IFD as follows:

```
$ dmpf skip=3472 count=20 bs=2 endian=0 hex=1 ~/ORF.ORF                                  Bash
   0xd90     3472: OLYMPUS_II._.__.._..              ->  4c4f 4d59 5550   53 4949    3
                                                                                  II  unknow
```

The offsets in this IFD are relative to the start of the MakeNote.

Reading this is easy:

```
    } else if ( image_.maker_ == kOlym ) {                                          C++
        Io     io(io_,offset,count);
        TiffImage makerNote(io,image_.maker_);
        makerNote.start_ = 12  ; // "OLYMPUS\0II\0x3\0x0"E#
        makerNote.valid_ = true; // Valid without magic=42
        makerNote.accept(visitor,makerDict());
    }
```

We treat it has a TiffImage (although invalid), set the start_ and valid_ variables and `accept()` parses the makernote effortlessly.

One of the interesting features of the ORF is the use of Tag Type IFD. These are used to introduce more families of data for ImageProcession, FocalInformation and other collections. Each of these IFDs requires a dictionary and these are defined in tvisitor.cpp. You recursively descend into those dictionaries as follows in IFD::accept():

```
    if ( type == kttIfd ) {                                                         C++
        for ( uint64_t i = 0 ; i < count ; i++ ) {
            offset = get4or8 (buff,0,i,endian);
            IFD(image_,offset,false).accept(visitor,ifdDict(image_.maker_,tag,makerDict(
        }
    } else switch ( tag ) {
        case ktGps       : IFD(image_,offset,false).accept(visitor,gpsDict );break;
        case ktExif      : IFD(image_,offset,false).accept(visitor,exifDict);break;
        case ktMakerNote :         visitMakerNote(visitor,buff,count,offset);break;
        default          : /* do nothing                                    */;break;
    }
```

The appropriate dictionary is selected with the code:

```C++
TagDict& ifdDict(maker_e maker,uint16_t tag,TagDict& makerDict)
{
    TagDict& result = makerDict ;
    if ( maker == kOlym ) switch ( tag ) {
        case 0x2010 : result = olymEQDict ; break;
        case 0x2020 : result = olymCSDict ; break;
        case 0x2030 : result = olymRDDict ; break;
        case 0x2031 : result = olymR2Dict ; break;
        case 0x2040 : result = olymIPDict ; break;
        case 0x2050 : result = olymFIDict ; break;
        case 0x3000 : result = olymRoDict ; break;
        default     : /* do nothing */    ; break;
    }
    return result;
}
```

There are many tags defined for the ORF file in Exiv2. Only a few have been defined in tvisitor.cpp for illustration purposes. To see unknown tags:

```Bash
.../book $ build/tvisitor -pU ~/ORF.ORF
...
      STRUCTURE OF  FILE (II): /Users/rmills/ORF.ORF:3472->1452144
           116 | 000000 Exif.OlympusEQ.Version       | UNDEFINED |         4 |
           128 | 0x0100 Exif.OlympusEQ.CameraType    |     ASCII |         6 |       4304
           140 | 0x0101 Exif.OlympusEQ.SerialNumber  |     ASCII |        32 |       4310
           152 | 0x0102 Exif.OlympusEQ.InternalSer.. |     ASCII |        32 |       4342
           164 | 0x0103 Exif.OlympusEQ.0x103         |  RATIONAL |         1 |       4376
           176 | 0x0104 Exif.OlympusEQ.0x104         |      LONG |         1 |
           188 | 0x0201 Exif.OlympusEQ.0x201         |     UBYTE |         6 |       4394
           200 | 0x0202 Exif.OlympusEQ.0x202         |     ASCII |        32 |       4400
           212 | 0x0203 Exif.OlympusEQ.0x203         |     ASCII |        32 |       4432
...
```

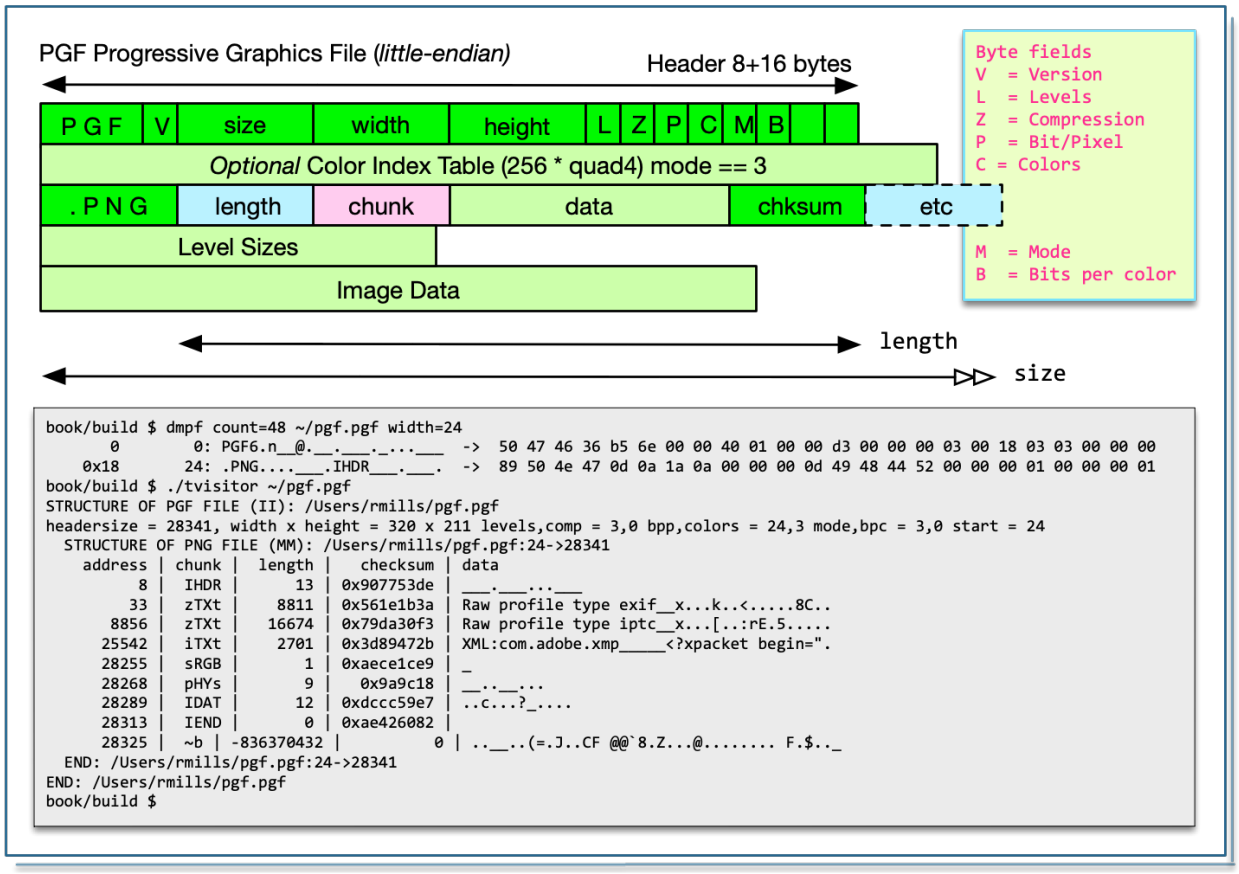
TOC

Pentax Raw

PEF is Tiff. I haven't found anything special about the PEF format. Of course, it has code for the Pentax MakerNote and that code is shared with some AVIF files in which the maker is "Ricoh".

TOC

PGF Progressive Graphics File



The PGF website is https://www.libpgf.org

This file format was introduced in 2000 at the same time as JP2000 with the intention of replacing JPEG. Neither PGF nor JP2000 have been successful in their aims. While both are technically superior to JPEG, the market as remained loyal to JPEG. I suspect that is because few Camera manufacturers ship products that support this format.

The file format is little-endian. Curiously, the metadata is embedded as a PNG which is big-endian.

The code to validate the file is simple:

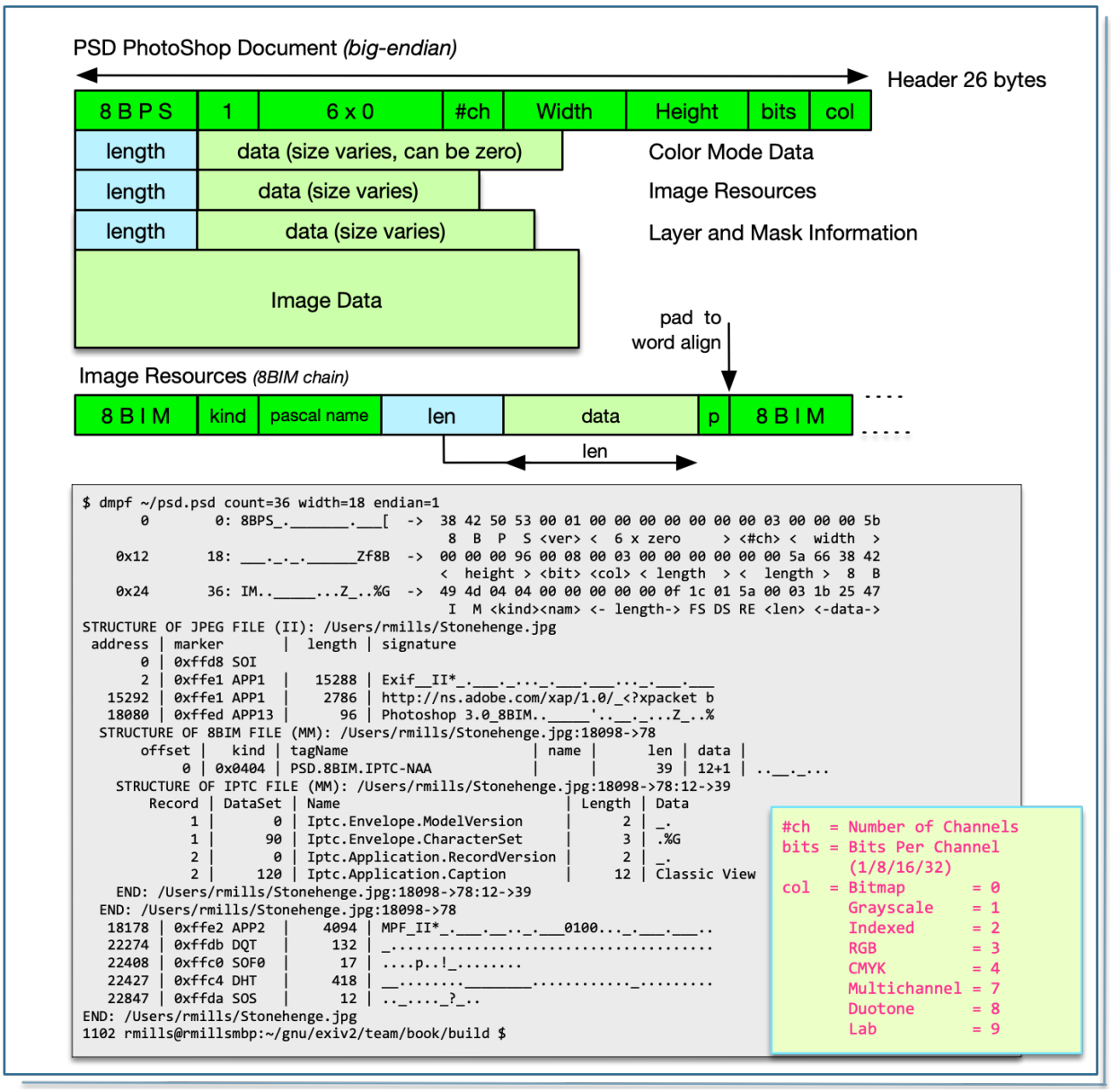
```C++
bool valid()
{
    if ( !valid_ ) {
        endian_ = keLittle ;
        IoSave   restore(io(),0);
        start_  = 8+16    ;
        DataBuf  h(start_);
        io_.read(h);

        if ( h.begins("PGF")   ) {
            start_  = 8+16;
            format_ = "PGF";
            valid_  = true;
        }
        headersize_ = getLong(h,  4,endian_);
        width_      = getLong(h,8 +0,endian_);
        height_     = getLong(h,8 +4,endian_);
        levels_     = getByte(h,8 +8);
        comp_       = getByte(h,8 +9);
        bpp_        = getByte(h,8+10);
        colors_     = getByte(h,8+11);
        mode_       = getByte(h,8+12);
        bpc_        = getByte(h,8+13);
        if ( mode_ == 3 ) start_ += 4*256; // start following color table
    }
    return valid_ ;
}
```

The visitor code is also straightforward:

```C++
void PGFImage::accept(Visitor& visitor)
{
    // Ensure that this is the correct image type
    if (!valid()) {
        std::ostringstream os ; os << "expected " << format_ ;
        Error(kerInvalidFileFormat,io().path(),os.str());
    }
    std::string msg = stringFormat("headersize = %d, width x height = %d x %d levels,cor
                      "bpp,colors = %d,%d mode,bpc = %d,%d start = %d"
                      ,headersize_,width_,height_,levels_,comp_,bpp_,colors_,mode_,bpc_,
    visitor.visitBegin(*this,msg); // tell the visitor
    PngImage(io(),start_,this->headersize_).accept(visitor);
    visitor.visitEnd  (*this); // tell the visitor
}
```

TOC

PSD PhotoShop Document



Adobe publish the following: https://www.adobe.com/devnet-apps/photoshop/fileformatashtml/.

PSDImage::valid() is straightforward:

```C++
bool PSDImage::valid()
{
    if ( !valid_ ) {
        endian_ = keBig ;
        IoSave   restore(io(),0);
        DataBuf  h(4); io_.read(h);
        uint16_t version = io_.getShort(endian_);

        if ( h.begins("8BPS") && version == 1 && io_.getLong(endian_) == 0 && io_.getSh
            start_  = 26;
            format_ = "PSD";
            valid_  = true;
        }
        ch_     = io_.getShort(endian_);
        width_  = io_.getLong (endian_);
        height_ = io_.getLong (endian_);
        bits_   = io_.getShort(endian_);
        col_    = io_.getLong (endian_);
        header_ = " address | length | data";
    }
    return valid_ ;
}
```

PSDImage::accept() is easy. As the header has been validated, all it has to do is to run the link-list of 3 resources (Color, Image and Layer/Mask). However, because the Metadata is all stored in the "Image Resources", I also navigate that structure and call visit8BIM().

There is a 'name' in the data structure which is stored as a Pascal string which is an array of up to 257 bytes. The first byte in the length of the string. The string is not null terminated. The string "\0\0" is the empty string. I think this is a software relic as my test file has the empty string for every 8BIM record.

I've chosen to treat the 8BIM chain as a file type. You will never find this as a file on your computer, however you will find this record type in JPEG files in App13 with signature ***PhotoShop 3.0***.

```C++
void PSDImage::accept(Visitor& visitor)
{
    // Ensure that this is the correct image type
    if (!valid()) {
        std::ostringstream os ; os << "expected " << format_ ;
        Error(kerInvalidFileFormat,io().path(),os.str());
    }
    std::string msg = stringFormat("#ch = %d, width x height = %d x %d, "
        "bits/col = %d/%d",ch_,width_,height_,bits_,col_);
    visitor.visitBegin(*this,msg); // tell the visitor

    IoSave restore(io_,0) ;
    DataBuf  lBuff(4);
    uint64_t address = start_ ;
    for ( uint16_t i = 0 ; i <=2 ; i++ ) {
        io().seek(address);
        uint32_t length = io_.getLong(endian());
        visitor.visitResource(io_,*this,address);
        Io    bim(io_,address+4,length);
        C8BIM c8bim(bim);
        c8bim.accept(visitor);
        address += length + 4 ;
    }

    visitor.visitEnd  (*this); // tell the visitor
}
```

ReportVisitor::visitResource() is called 3 times and reports the address,length and data.

```C++
void ReportVisitor::visitResource(Io& io,Image& image,uint64_t address)
{
    IoSave   restore(io,address);
    uint32_t length = io.getLong(image.endian());
    DataBuf  buff(length > 40 ? 40 : length);
    io.read(buff);
    out() << indent() << stringFormat("%8d | %6d | ",address,length) << buff.binaryToStr
}
```

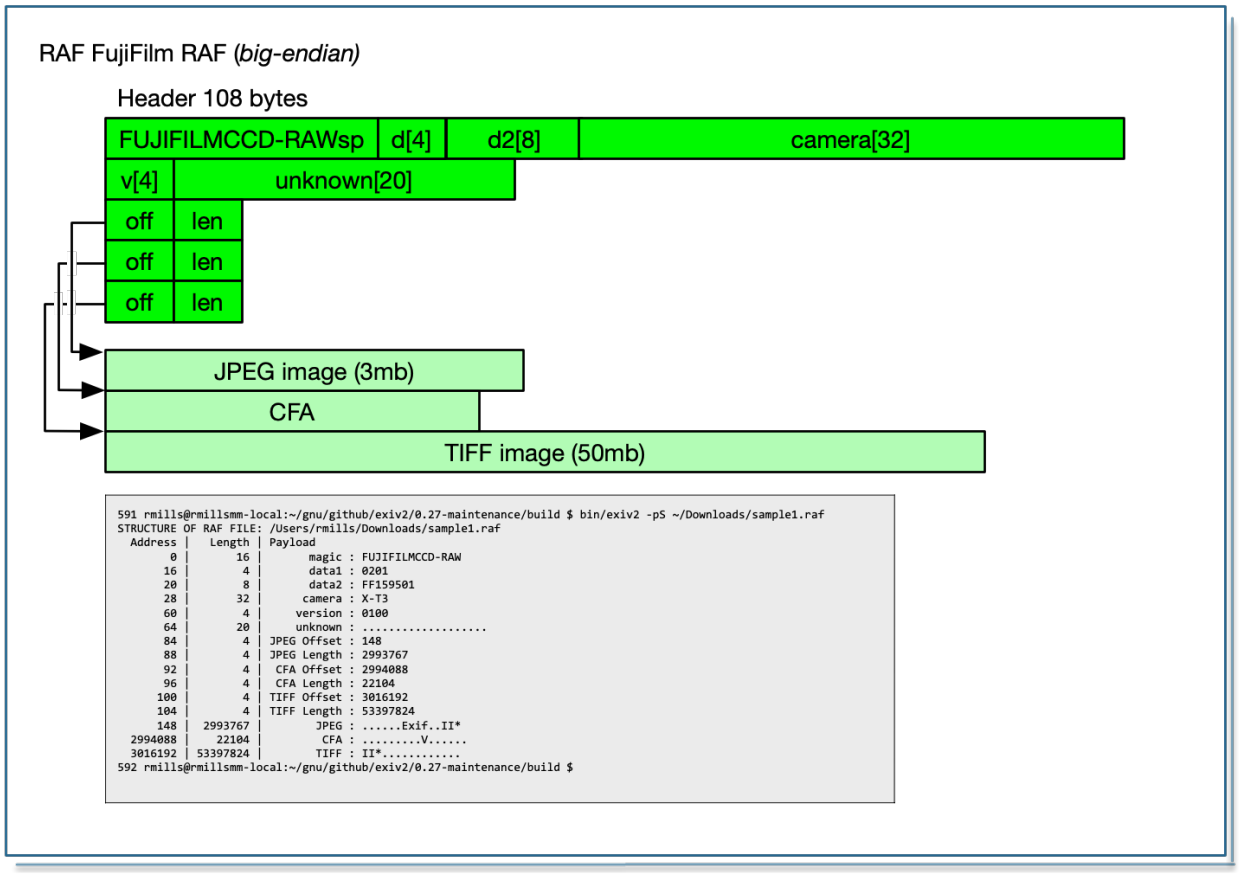
And to complete the story, the reporter for 8BIM is quite simple.

```C++
void ReportVisitor::visit8BIM(Io& io,Image& image,uint32_t offset
                ,uint16_t kind,uint32_t len,uint32_t data,uint32_t pad,DataBuf& b)
{
    std::string tag = ::tagName(kind,psdDict,40,"PSD");
    if ( printTag(tag) ) {
        out() << indent()
              << stringFormat("   %8d | %#06x | %-28s | %4d | %2d+%1d | "
                              ,offset,kind,tag.c_str(),len,data,pad)
              << b.binaryToString()
              << std::endl;
    }
}
```

I haven't bothered to implement options -pX (XMP), -pC (ICC Color Profile) or -pI (IPTC) although it's very simple to implement.

TOC

RAF Fujifilm RAW



I found this useful description: https://libopenraw.freedesktop.org/formats/raf. I don't recognise the format of the embedded CFA. I believe CFA is Color Filter Array.

Most of the metadata is contained in the embedded JPEG. However there is metadata in the embedded TIFF. This is discussed here: https://github.com/Exiv2/exiv2/issues/1402.

The MakerNote in the embedded JPEG in a RAF has a 12 byte header followed by an IFD. The 12 bytes header is the ascii string FUJIFILM followed by the bytes 0x0c 0 0 0. Perhaps it's a coincidence that that 0x0c00000000 is bigEndian '12'. It's possible that the header is "FUJIFILM"long and long is the offset to the IFD. As RAF is a big endian file, that's possible. The code in both Exiv2 and tvisitor however simply skips the 12 byte header and reads the IFD.

TOC

RW2 Panasonic RAW

There is a discussion of Raw Image Formats here: https://en.wikipedia.org/wiki/Raw_image_format

RW2 is effectively Tiff, however Panasonic:

1. Put their keys into the top level dictionary.
2. Use 0x55 = 'U' = 85 in the 'magic' header.

```Bash
914 rmills@rmillsmbp:~/gnu/exiv2/team/book/build $ dmpf count=20 ~/Downloads/RAW_P
       0        0: IIU_.___#_._._.___02              ->  49 49 55 00 08 00 00 00 23 00 0
915 rmills@rmillsmbp:~/gnu/exiv2/team/book/build $
```

We deal with those differences in TiffImage::valid() as follows:

```C++
    valid_ =  (magic_==42||magic_==43||magic_==85) && (c == C) && (c=='I'||c=='M')
    // Panosonic have augmented tiffDict with their keys
    if ( magic_ == 85 ) {
        setMaker(kPano);
        for ( TagDict::iterator it = panoDict.begin() ; it != panoDict.end() ; it++ ) {
            if ( it->first != ktGroup ) {
                tiffDict[it->first] = it->second;
            }
        }
    }
```

TOC

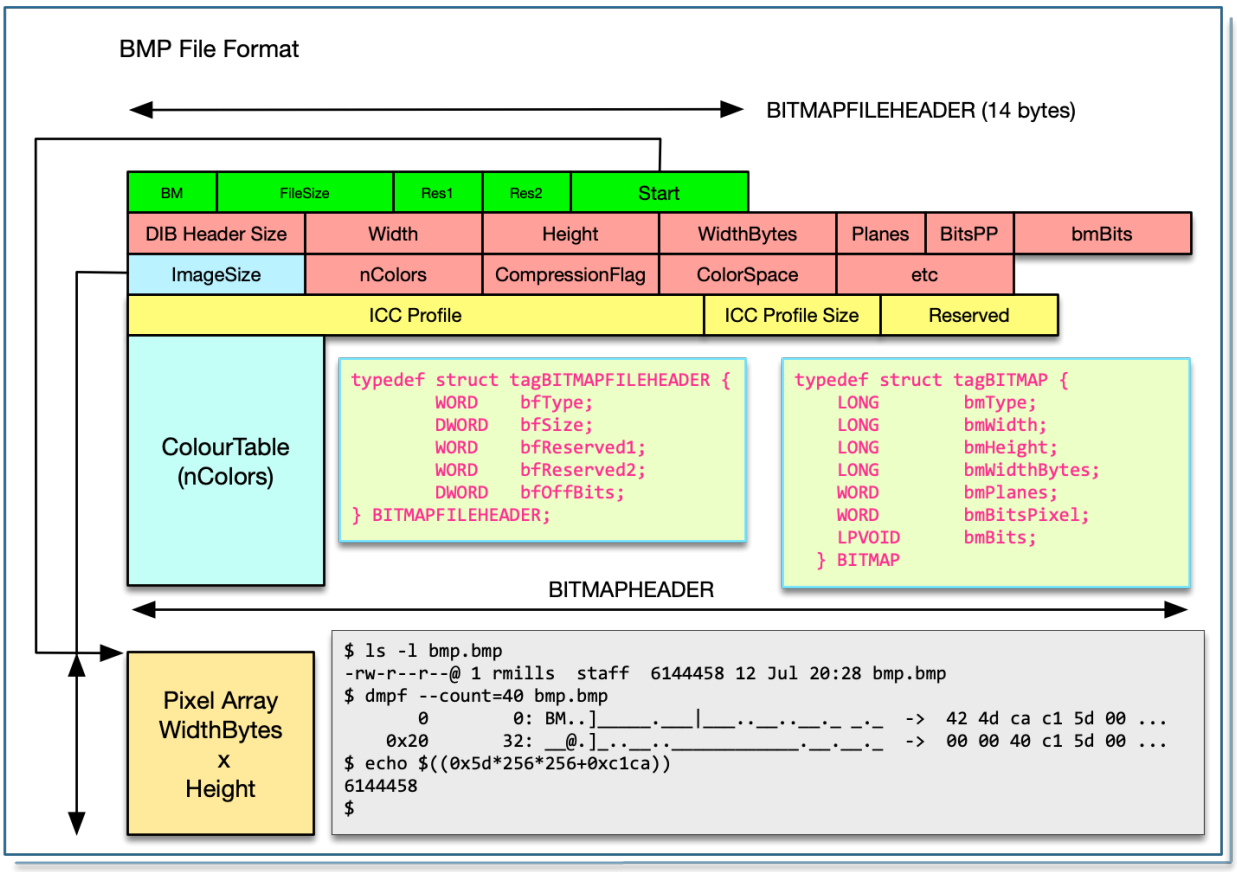
TGA Truevision Targa

The TGA file format is documented here:http://www.ludorg.net/amnesia/TGA_File_Format_Spec.html

There is no Exiv2, xmp, ICC or IPTC metadata in a TGA file.

TOC

BMP Windows Bitmap



The Windows Bitmap Format "Bimp" has been around in Windows forever. It may even have been in Windows 1.0. For sure, it was there when I started to working with Windows 3.0 in 1991. It's not a container. It's a few fields of data and lots of pixels!

BMP is often referred to as a "device independant bitmap" because it's not designed for any physical device. It is the responsibility of the device or printer driver to render the image. Windows device drivers are required to implement the GDI (Graphical Device Interface).

The first 4 bytes of the BITMAP (following the 14 byte file BITMAPFILE header) is the length of BITMAP. As well as being useful for navigating the file, this is effectively the BITMAP version. In the Windows SDK, they call it bmType. The last time I looked at the structure of a BMP (Windows 95) it believe it was 42 bytes. On Windows 10 it's now 124.

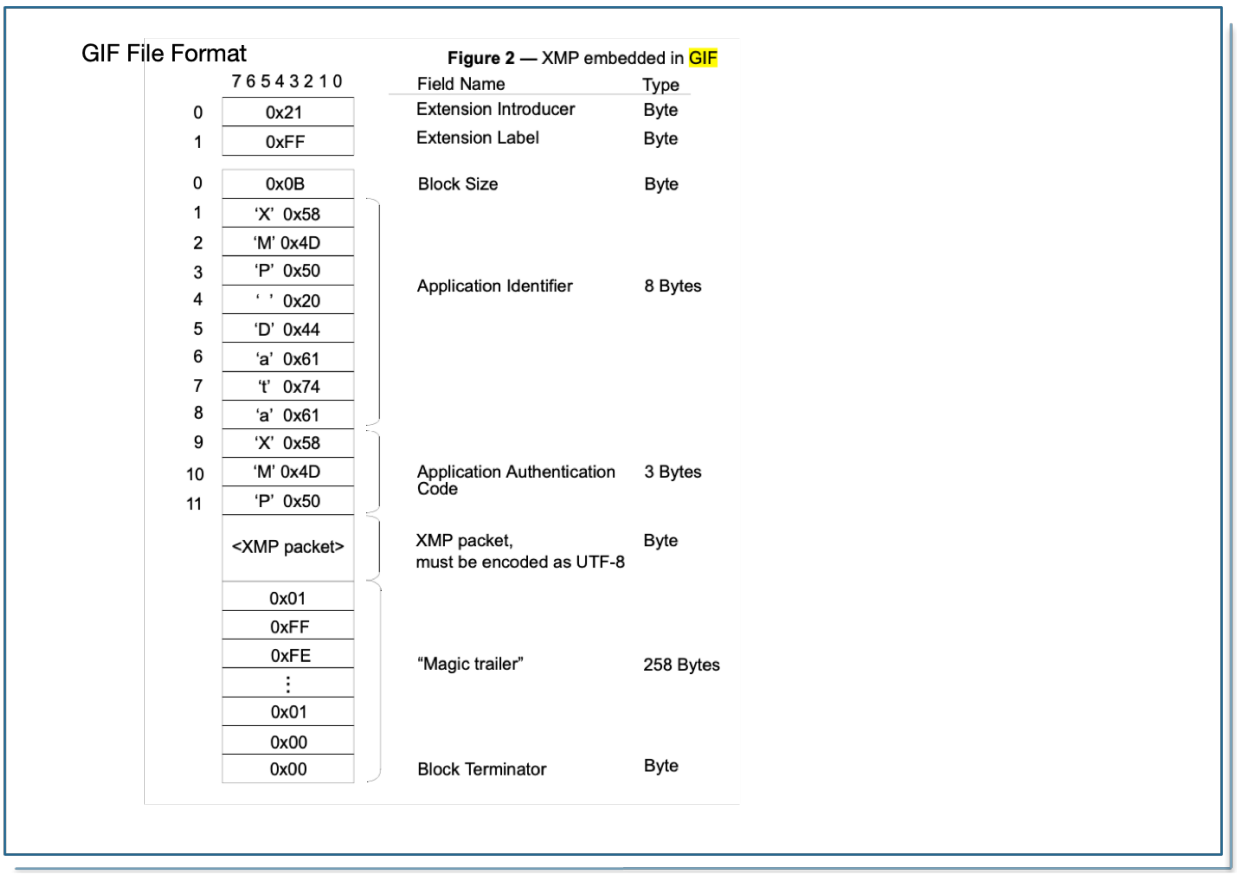
From a metadata standpoint, there's almost nothing interesting in a BMP. I was surprised to discover that the latest version (BITMAPV5) can store an ICC profile and has an alpha channel bitmask. Perhaps a future version will include XMP and Exif metadata.

A BMP can be rendered with transparency. The GDI has a method TransparentBlt() in which one colour value is defined to be "transparent". This method of supporting transparency is the responsibility of the application code and not defined in the file itself.

Before moving on from BMP, I'd like to say something about the flexibility of the BMP format. You can have different colours depths and the image can have indexed color. In this format, a table of up to 256 colours can be defined and the value of a pixel is the index and not the colour itself. It's interesting to see that Microsoft have been working with this for 40 years, occasionally upgrading, and have never broken backwards compatibility. Everybody would benefit from camera manufacturers adopting a similar approach to file formats.

TOC

GIF Graphics Image Format



Component	URL
Gif Specification	https://www.w3.org/Graphics/GIF/spec-gif89a.txt
LibGif	https://sourceforge.net/projects/giflib/
LibGif Man Pages	https://www.mankier.com/1/gifbuild
Adobe XMPsdk	https://github.com/adobe/XMP-Toolkit-SDK.git
WikiPedia GIF	https://en.wikipedia.org/wiki/GIF

I built GifLib 5.1.1 on macOS . 5.2.1 complained about the option linker option *-soname* and refused to link!

I followed the discussion on the WikiPedia site and created a 3x2 pixel gif with MSPaint on Windows-10 which I've inspected with the giflib/. The file gif.gif is in the book resources at: svn://dev.exiv2.org/svn/team/book

```Bash
.../book $ gifbuild -d -v gif.gif
#
# GIF information from gif.gif
screen width 2
screen height 3
screen colors 256
screen background 0
pixel aspect byte 0

screen map
    sort flag off
    rgb 000 000 000
    rgb 000 000 051
.... color table entries ....
    rgb 000 000 000
end

graphics control
    disposal mode 0
    user input flag off
    delay 0
    transparent index 252
end

image # 1
image left 0
image top 0
image bits 2 by 3 hex
9993
9300
fbe0
```

```Bash
.../book$ dmpf gif.gif
       0        0: GIF89a._._._______3__f__.__.__._  ->  47 49 46 38 39 61 02 00 03 00 f
                                                          G  I  F  8  9  a <-W-> <-H-> <
    0x20       32: +__+3_+f_+._+._+._U__U3_Uf_U._U.  ->  2b 00 00 2b 33 00 2b 66 00 2b 9
...
   0x300      768: ._____________!..__._,____._._.__.  ->  ff 00 00 00 00 00 00 00 00 00 0
   0x320      800: ._3M..`..._;                        ->  09 00 33 4d 9a 04 60 1f b8 80 0
```

XMP in Gif

This is supported by Gif89a files and documented by Adobe in XMPSpecificationPart3.pdf on page 17 of the 2010 edition. At present I don't have a sample GIF with embedded XMP. I don't know if Exiv2 supports GIF/XML.

TOC

SIDECAR Xmp Sidecars

Sidecar files are “Raw” XMP. They are used for several purposes such as to provide XMP support for files for which there is no define standard encoding. For example, if you use less common legacy formats such as Sun Raster, the simplest way to provide XMP support is to use a sidecar. Conventionally for a file such as foo.ras, sidecar will have the foo.xmp

Sidecar files are sometimes used to store application data. For example, the Adobe Camera Raw Convertor installs LensProfiles .lcp files in /Library/Application Support/Adobe/CameraRaw/LensProfiles/ (on macOS).

TOC

2 Metadata Standards

Exif is the most important of the metadata containers. However others exist and are supported by Exiv2:

Type	Definition	Comment
EXIF	EXchangeable Image Format	Japanese Electronic Industry Development Association Standard
IPTC	International Press Telecommunications Council	Press Industry Standard
Xmp	Extensible Metadata Platform	Adobe Standard
ICC	International Color Consortium	Industry Consortium for Color Handling Standards

TOC

2.1 Exif Metadata

EXIF = Exchangeable image file format. Exif is the largest and most commonly used metadata standard. The standard is defined by JEITA which is the Japanese Association of Camera Manufacturers. Exif metadata is embedded in almost all images captured by cameras, phones and other "smart" devices. Exif has tags for Maker, Model, Aperture and many other settings. Exiv2 supports the Exif 2.32 Standard. Exiv2 knows the definition of about 6000 tags. Exif however is not extensible. Over the years, tags have been added for topics such as GPS, Lens and Time Zone.

To enable the manufacturer to store both proprietary and non-standard data, the MakerNote Tag is defined. Usually the Manufacturer will write a TIFF Encoded record into the MakerNote. Exiv2 can reliably read and rewrite Manufacturer's MakerNotes. The implementation of this in Exiv2 is the outstanding work of Andreas Huggel.

In order to understand the Exif Standard, it's useful to understand its relationship with other standards:

Name	Description	URL
TIFF 6.0	Adobe Standard. It defines the structure of a Tiff File (the IFD) and tags for image properties such as ImageWidth and ImageHeight	TIFF6.pdf
TIFF-EP	ISO Standard. TIFF for Electronic Photographs extends TIFF 6 to provide tags for Photographs. For example: IPTC/NAA and ISOSpeedRatings	TAG2000-22_DIS12234-2.pdf
DNG	Adobe Standard. DNG extends TIFF-EP with tags for Camera Raw Processing. For example: CameraCalibration	dng_spec_1.5.0.0.pdf
Exif	JEITA Standard.	CIPA_DC_008_EXIF_2019.pdf
XMP	Adobe Standard. XMP is written in XML. For now, we're only concerned with embedding this in Tiff.	XMPSpecificationPart3.pdf
ICC Profile	International Color Consortium Standard. The specification defines both the ICC Profile Format and embedding standards.	ICC1v43_2010-12.pdf
IPTC	ITPC Standard.	IPTC-PhotoMetadata-201407_1.pdf

Exif Standard Tags

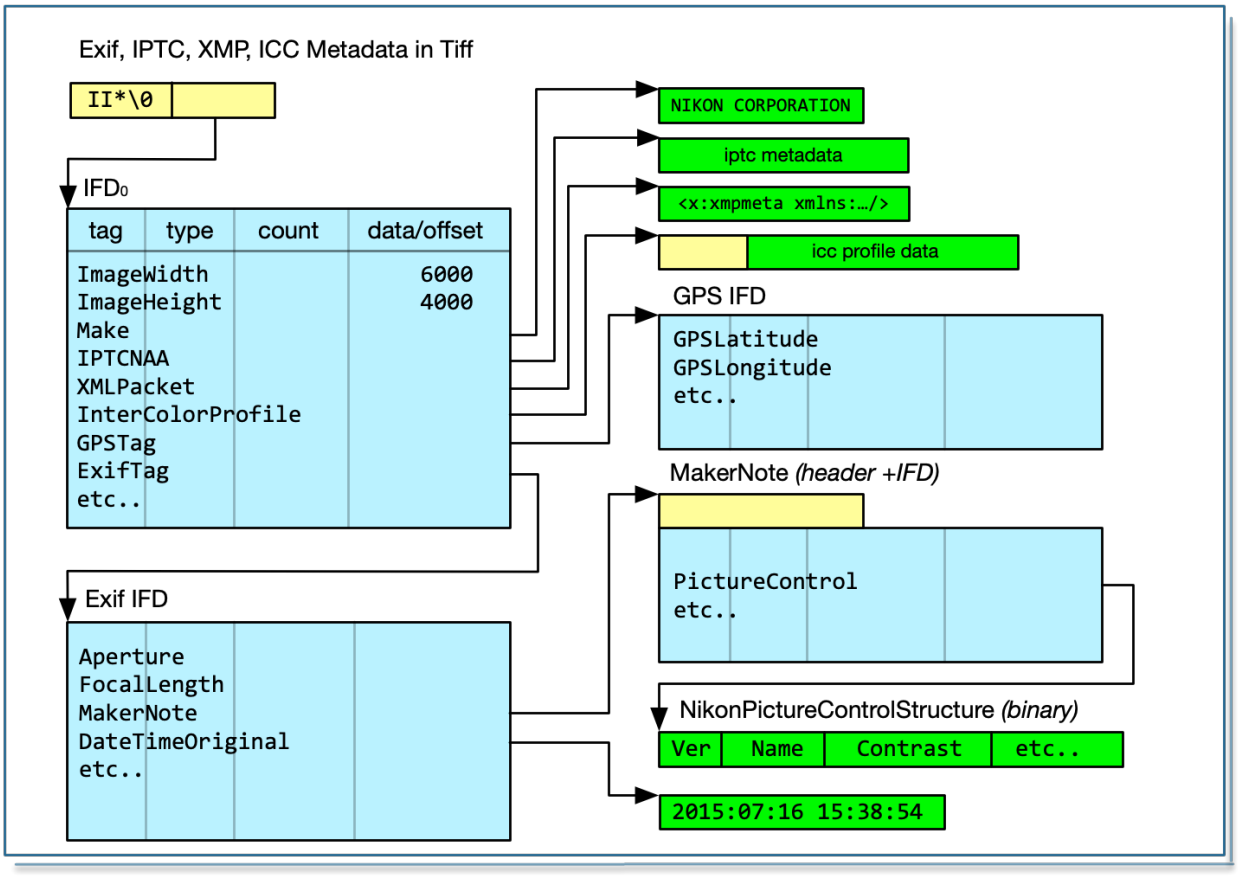
Specification	IFD	Section	Tag Examples
Exif 4.6.3	IFD0	Offset to Exif and GPS IFDs	ExifTag, GPSTag
Exif 4.6.4	IFD0	Image data structure Make Image data location Image data characteristics Other Tags	ImageWidth, ImageHeight Make StripOffsets, RowsPerStrip TransferFunction, WhitePoint ImageDescription, DateTime
ICC XMP TIFF-EP	IFD0	ICC Profile XMP IPTC/NAA *See* **ICC/XMP/IPTC** *below*	InterColorProfile XMLPacket IPTCNAA
Exif 4.6.5	Exif IFD	Exif Version Image Data Characteristics Image Configuration User or Manufacturer Information Related File Information Date and Time Picture Conditions Shooting Situation Other	ExifVersion ColorSpace, Gamma ComponentsConfiguration, CompressedBitsPerPixel UserComment, MakerNote *See* **MakerNote** *below* RelatedSoundFile DateTimeOriginal Aperture, FocalLength Temperature, CameraElevationAngle LensSpecification, CameraOwnerName
Exif 4.6.6	GPS IFD	GPS Data	GPSSatellites, GPSLatitude
Exif 4.6.7	Interop IFD	*See* **Interop** *below*	

Interop The Exif standard says: *The Interoperability structure of Interoperability IFD is same as TIFF defined IFD structure but does not contain the image data characteristically compared with normal TIFF IFD.*

MakerNote The Exif standard does not define the structure of the MakerNote. In practice, all manufacturers store their private data as a short header followed by an IFD or embedded TIFF file. The contents of the makernote headers are documented: https://exiv2.org/makernote.html. There is an index to every Exif (and IPTC and XMP and MakerNote) tag supported by Exiv2 at https://exiv2.org/metadata.html. For example the Canon MakerNote is documented: https://exiv2.org/tags-canon.html.

ICC/XMP/IPTC The tags InterColorProfile, XMLPacket and IPTCNAA are usually only found in Tiff files. Other formats such as JPEG, PNG, JP2 use different mechanisms to embed the data. How the data is embedded is defined in the ICC Profile Specification and discussed in the Image File Formats in Chapter 1.

Structure of Exif Metadata



Exif metadata is stored as an embedded TIFF. So, every tag has an array of values. The array is homogeneous. Every element of the array is of the same type. So, there can be an array of ASCII values *(for strings)*, or an array of Shorts *(for image dimensions)*. Foreign data such as IPTC, ICC Profiles and MakerNotes are typically an array of "UNDEFINED" which are binary data.

There are 3 types of single-byte arrays in Exif. An array of ASCII values should by 7-bit ascii values with a trailing null. An array of UNDEFINED is usually use to define binary data. An array of BYTE values is a byte-stream of uint8_t and typically used by XMPPacket to store XML.

Character Set Encoding in Strings

There are only 3 types of string in Exif which use CharSet encoding. They are UserComment, GPSAreaInformation and GPSProcessingMethod

```
1 $ taglist ALL | csv - | grep '\[Comment\]'
2 [Photo.UserComment] [37510] [0x9286]    [Photo] [Exif.Photo.UserComment]    [Comment]
3 [GPSInfo.GPSProcessingMethod]   [27]    [0x001b]    [GPSInfo]   [Exif.GPSInfo.GPSProcess
4 [GPSInfo.GPSAreaInformation]    [28]    [0x001c]    [GPSInfo]   [Exif.GPSInfo.GPSAreaInf
5 $
```

The structure of those tags is defined on page34 of the Exiv2-2 Specification. Three standards are supported by the Specification and they are "ASCII, JIS and UNICODE". Provision is a made for "Undefined" which presumably leaves it to the application to interpret the data.

These tags are discussed and explained in the Exiv2 man page. The code in tvisitor.cpp does not deal with CharSet decoding and reports the binary values of the bytes.

The tag XMLPacket is normally an array of BYTE values. The encoding of such a string defaults to UTF-8 and can be defined in the XML Processing Instruction. For an Exif/Tiff perspective, the count is the number of bytes used to store the string. The length of the decoded string may differ.

Inspecting the Values of Exif Data

Before we get into the Exiv2 code, let's look at the simpler python TIFF/Exif library.
https://github.com/Moustikitos/tyf

You will also need to install PIL:

```Bash
$ sudo python3 -m pip install Pillow
$ git clone https://github.com/Moustikitos/tyf
$ cd tyf
$ sudo python3 setup.py install
```

This is a library and I've constructed a program to reveal the Exif metadata.

```Python
#!/usr/bin/env python3

import Tyf
import os
import sys

import urllib.request
from io  import BytesIO
from PIL import Image

##
#
def dumpTags(ifd):
    bDumpTags    = True
    bGenerateMap = False

    if bDumpTags:
        for tag in ifd:
            V=tag[1]
            v=str(V)
            if type(V)==type(''):
                v='"'+v+'"'
            if len(v) > 30:
               v = v[0:26] + '.. '
            t=str(type(V))
            t=t[8:len(t)-2]
            if t == 'bytes':
                t=str(len(V)) + ' ' + t
            elif t == 'str':
                t=str(len(V))
            if len(t) > 30:
                t = t[0:26]+'.. '

            t='('+t+')'
```

```
            t= ( +t+ )

            print('%s -> %s %s' % ( tag[0], v , t)  )

##
#
def main(argv):
    """main - main program of course"""

    image = Tyf.open(argv[1])
    # help(jpg)

    if str(type(image)) == "<class 'Tyf.TiffFile'>":
        dumpTags(image[0])
    elif str(type(image)) == "<class 'Tyf.JpegFile'>":
        dumpTags(image.ifd0)
    else:
        print("unknown image type " + str(type(image)))

if __name__ == '__main__':
    main(sys.argv)

# That's all Folks
##
```

Download from https://clanmills.com/Stonehenge.jpg and https://clanmills.com/Stonehenge.tif

```Bash
$ ~/bin/mdump.py  ~/Stonehenge.jpg
Make -> "NIKON CORPORATION" (17)
Model -> "NIKON D5300" (11)
Orientation -> 1 (int)
XResolution -> 300.0 (float)
YResolution -> 300.0 (float)
ResolutionUnit -> 2 (int)
Software -> "Ver.1.00 " (9)
DateTime -> 2015-07-16 20:25:28 (datetim
YCbCrPositioning -> 1 (int)
Exif IFD -> 222 (int)
...
MakerNote -> b'Nikon\x00\x02\x11\x00\x00'
UserComment ->
SubsecTime -> "00" (2)
SubsecTimeOriginal -> "00" (2)
SubsecTimeDigitized -> "00" (2)
FlashpixVersion -> b'0100' (4 bytes)
ColorSpace -> 1 (int)
PixelXDimension -> 6000 (int)
PixelYDimension -> 4000 (int)
Interoperability IFD -> 4306 (int)
SensingMethod -> 2 (int)
FileSource -> b'\x03' (1 bytes)
SceneType -> b'\x01' (1 bytes)
CFAPattern -> b'\x02\x00\x02\x00\x00\x01'
CustomRendered -> 0 (int)
ExposureMode -> 0 (int)
...
GPSLatitudeRef -> 1 (int)
GPSLatitude -> 51.17828166666666 (float)
GPSLongitudeRef -> -1 (int)
GPSLongitude -> 1.8266399999999998 (floa
GPSAltitudeRef -> -1 (int)
GPSAltitude -> 97.0 (float)
GPSTimeStamp -> 14:38:55 (datetime.time)
GPSSatellites -> "09" (2)
GPSMapDatum -> "WGS-84          " (16)
GPSDateStamp -> 2015-07-16 00:00:00 (dat
...
```

```
$ ~/bin/mdump.py  ~/Stonehenge.tif
ImageLength -> 1 (int)
BitsPerSample -> (8, 8, 8, 8)
Compression -> 1 (int)
PhotometricInterpretation -> 2
FillOrder -> 1 (int)
ImageDescription -> "Classic V"
Make -> "NIKON CORPORATION" (1
Model -> "NIKON D5300" (11)
StripOffsets -> 901 (int)
Orientation -> 1 (int)

ExposureTime -> 0.0025 (float)
FNumber -> 10.0 (float)
ExposureProgram -> 0 (int)
ISOSpeedRatings -> 200 (int)
ExifVersion -> b'0230' (4 byte
DateTimeOriginal -> 2015-07-16
DateTimeDigitized -> 2015-07-1
ComponentsConfiguration ->
CompressedBitsPerPixel -> 2.0
ExposureBiasValue -> (0, 1) (t
MaxApertureValue -> 4.3 (float
MeteringMode -> 5 (int)
LightSource -> 0 (int)
 Flash -> 16 (int)
FocalLength -> 44.0 (float)
UserComment ->

WhiteBalance -> 0 (int)
DigitalZoomRatio -> 1.0 (float
FocalLengthIn35mmFilm -> 66 (i
SceneCaptureType -> 0 (int)
GainControl -> 0 (int)
Contrast -> 0 (int)
Saturation -> 0 (int)
Sharpness -> 0 (int)
SubjectDistanceRange -> 0 (int
ImageUniqueID -> "090caaf..."
GPSLatitudeRef -> 1 (int)
   GPSLatitude -> 51.178280555555
```

Data's similar. The order is different. Good news is that the commands ***$ exiv2 -pe ~/Stonehenge.jpg*** and ***$ exiv2 -pe ~/Stonehenge.tif*** produce similar data in the same order. We'd hope so as both commands are reading the same embedded Exif metadata. The way in which the Exif is embedded in Tiff and JPG is different, however the Exif metadata is effectively the same.

TOC

2.2 XMP Metadata

XMP is an Adobe initiative to provide a comprensive and eXtendable Metadata frame to a wide range of documents.

You can create Bag, Seq or Struct of metadata. An “XmpBag” is a set of key/value pairs and are represented by XML attributes. An “XmpSeq” is a an array of metadata similar to a JavaScript or Python Array. It’s represented by an XML list and can be accessed by index. An XmpStruct is a set of keys to trees of metadata rathen like a JavaScript or Python Object or Python.

Here are a couple of discussions about XMP on GitHub and Redmine.
https://github.com/Exiv2/exiv2/issues/1254 and https://dev.exiv2.org/boards/3/topics/2016.

Exiv2 provides a veneer over Adobe XMPsdk that makes it quite easy to work with XMP. As XMP is eXtensible, you are more-or-less free to create arbitrary trees of metadata which conform to the RDF schema. I strongly recommend however that applications emulate the XMP generated by Adobe Applications as that promotes better interoperability.

Exiv2 is not a metadata policeman. You are provided with tools to modify metadata. ***As Jabba said in Star Wars:*** *“With great power comes great responsibility.”*. Use the tools wisely. Learn the ways of the force!

I have taken the XMP example from this web-site and simplified it a little into the file ***xmp.xmp***
https://en.wikipedia.org/wiki/Extensible_Metadata_Platform

```
1 <?xpacket begin="?" id="W5M0MpCehiHzreSzNTczkc9d"?>                    Markup
2  <x:xmpmeta xmlns:x="adobe:ns:meta/" x:xmptk="Adobe XMP Core 5.4-c002 1.000000, 0000/00/
```

```
   <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
      <rdf:Description rdf:about=""
            xmlns:xmp="http://ns.adobe.com/xap/1.0/">
         <xmp:CreatorTool>Picasa</xmp:CreatorTool>
      </rdf:Description>
      <rdf:Description rdf:about=""
            xmlns:mwg-rs="http://www.metadataworkinggroup.com/schemas/regions/"
            xmlns:stDim="http://ns.adobe.com/xap/1.0/sType/Dimensions#"
            xmlns:stArea="http://ns.adobe.com/xmp/sType/Area#">
         <mwg-rs:Regions rdf:parseType="Resource">
            <mwg-rs:AppliedToDimensions rdf:parseType="Resource">
               <stDim:w>912</stDim:w>
               <stDim:h>687</stDim:h>
               <stDim:unit>pixel</stDim:unit>
            </mwg-rs:AppliedToDimensions>
            <mwg-rs:RegionList>
               <rdf:Bag>
                  <rdf:li rdf:parseType="Resource">
                     <mwg-rs:Area rdf:parseType="Resource">
                        <stArea:x>0.680921052631579</stArea:x>
                        <stArea:y>0.3537117903930131</stArea:y>
                        <stArea:h>0.4264919941775837</stArea:h>
                        <stArea:w>0.32127192982456143</stArea:w>
                     </mwg-rs:Area>
                  </rdf:li>
               </rdf:Bag>
            </mwg-rs:RegionList>
         </mwg-rs:Regions>
      </rdf:Description>
   </rdf:RDF>
 </x:xmpmeta>
<?xpacket end="w"?>
```

You can add this directly into a file as follows:

```Bash
$ curl -LO http://clanmills.com/Stonehenge.jpg
$ cat xmp.xmp | exiv2 -iX- ~/Stonehenge.jpg
$ exiv2 -px Stonehenge.jpg
980 rmills@rmillsmbp:~/temp $ exiv2 -px Stonehenge.jpg
Xmp.xmp.CreatorTool                              XmpText     6  Picasa
Xmp.mwg-rs.Regions                               XmpText     0  type="Struct"
Xmp.mwg-rs.Regions/mwg-rs:AppliedToDimensions XmpText     0  type="Struct"
Xmp.mwg-rs.Regions/mwg-rs:AppliedToDimensions/stDim:w XmpText     3  912
Xmp.mwg-rs.Regions/mwg-rs:AppliedToDimensions/stDim:h XmpText     3  687
Xmp.mwg-rs.Regions/mwg-rs:AppliedToDimensions/stDim:unit XmpText     5  pixel
Xmp.mwg-rs.Regions/mwg-rs:RegionList             XmpText     0  type="Bag"
Xmp.mwg-rs.Regions/mwg-rs:RegionList[1]          XmpText     0  type="Struct"
Xmp.mwg-rs.Regions/mwg-rs:RegionList[1]/mwg-rs:Area XmpText     0  type="Struct"
Xmp.mwg-rs.Regions/mwg-rs:RegionList[1]/mwg-rs:Area/stArea:x XmpText    17  0.680921052(
Xmp.mwg-rs.Regions/mwg-rs:RegionList[1]/mwg-rs:Area/stArea:y XmpText    18  0.353711790
Xmp.mwg-rs.Regions/mwg-rs:RegionList[1]/mwg-rs:Area/stArea:h XmpText    18  0.426491994
Xmp.mwg-rs.Regions/mwg-rs:RegionList[1]/mwg-rs:Area/stArea:w XmpText    19  0.321271929
```

I find the structure easier to understand in JSON, which can be generated with the command ***$ exiv2json***

Stonehenge.jpg

```JavaScript
{
  "Xmp": {
    "xmp": {
      "CreatorTool": "Picasa"
    },
    "mwg-rs": {
      "Regions": {
        "mwg-rs": {
          "AppliedToDimensions": {
            "stDim": {
              "w": "912",
              "h": "687",
              "unit": "pixel"
            }
          },
          "RegionList": [
            {
              "mwg-rs": {
                "Area": {
                  "stArea": {
                    "x": "0.680921052631579",
                    "y": "0.3537117903930131",
                    "h": "0.4264919941775837",
                    "w": "0.32127192982456143"
            } } } }
          ]
        } }
    },
    "xmlns": {
      "AppliedToDimensions": "",
      "Area": "",
      "Regions": "",
      "mwg-rs": "http:\/\/www.metadataworkinggroup.com\/schemas\/regions\/",
      "xmp": "http:\/\/ns.adobe.com\/xap\/1.0\/"
    }
  }
}
```

You create XMP metadata with the syntax:

```Bash
$ exiv2 -M'set Xmp.namespace.Key value' path
```

Adobe XMPsdk isn't easy to understand. As I have never used it outside of Exiv2, my knowledge is limited. Exiv2 however enables you to insert, modify and delete simple values, Seq and Struct objects and Bags. You can create this XMP structure about using the Exiv2 command-line program as follows:

Step 1 Get an image and delete all XMP metadata:

```Bash
918 rmills@rmillsmbp:~/temp $ curl -LO http://clanmills.com/Stonehenge.jpg
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 6599k  100 6599k    0     0  1806k      0  0:00:03  0:00:03 --:--:-- 1806k
919 rmills@rmillsmbp:~/temp $ exiv2 -pX Stonehenge.jpg | xmllint -pretty 1 -
<?xml version="1.0"?>
<?xpacket begin="" id="W5M0MpCehiHzreSzNTczkc9d"?>
<x:xmpmeta xmlns:x="adobe:ns:meta/" x:xmptk="XMP Core 4.4.0-Exiv2">
  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <rdf:Description xmlns:xmp="http://ns.adobe.com/xap/1.0/" xmlns:dc="http://purl.org/
      <dc:description>
        <rdf:Alt>
          <rdf:li xml:lang="x-default">Classic View</rdf:li>
        </rdf:Alt>
      </dc:description>
    </rdf:Description>
  </rdf:RDF>
</x:xmpmeta>
<?xpacket end="w"?>
920 rmills@rmillsmbp:~/temp $ exiv2 -dX Stonehenge.jpg
921 rmills@rmillsmbp:~/temp $ exiv2 -pX Stonehenge.jpg | xmllint -pretty 1 -
-:1: parser error : Document is empty

^
925 rmills@rmillsmbp:~/temp $
```

Step 2 Create a simple XMP Property

```Bash
$ exiv2 -M'set Xmp.xmp.CreatorTool Picasa' Stonehenge.jpg
$ exiv2 -pX Stonehenge.jpg | xmllint -pretty 1 -
<?xml version="1.0"?>
<?xpacket begin="" id="W5M0MpCehiHzreSzNTczkc9d"?>
<x:xmpmeta xmlns:x="adobe:ns:meta/" x:xmptk="XMP Core 4.4.0-Exiv2">
  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <rdf:Description xmlns:xmp="http://ns.adobe.com/xap/1.0/" rdf:about="" xmp:CreatorT
  </rdf:RDF>
</x:xmpmeta>
<?xpacket end="w"?>
```

Step 3 Create nested Structures for Regions and Region:

```Bash
$ exiv2 -M'set Xmp.mwg-rs.Regions XmpText type=Struct' \
        -M'set Xmp.mwg-rs.Regions/mwg-rs:AppliedToDimensions XmpText type=Struct' \
        -M'set Xmp.mwg-rs.Regions/mwg-rs:AppliedToDimensions/stDim:w 912' Stonehenge.jpg
$ exiv2 -pX Stonehenge.jpg | xmllint -pretty 1 -
<?xml version="1.0"?>
<?xpacket begin="" id="W5M0MpCehiHzreSzNTczkc9d"?>
<x:xmpmeta xmlns:x="adobe:ns:meta/" x:xmptk="XMP Core 4.4.0-Exiv2">
  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <rdf:Description xmlns:xmp="http://ns.adobe.com/xap/1.0/" xmlns:mwg-rs="http://www.r
      <mwg-rs:Regions rdf:parseType="Resource">
        <mwg-rs:AppliedToDimensions stDim:w="912"/>
      </mwg-rs:Regions>
    </rdf:Description>
  </rdf:RDF>
</x:xmpmeta>
<?xpacket end="w"?>
$
```

Step 4 Create a bag with Struct of Areas:

```Bash
$ exiv2 -M'set Xmp.mwg-rs.Regions/mwg-rs:RegionList  XmpText type="Bag"' Stoneheng
$ exiv2 -M'set Xmp.mwg-rs.Regions/mwg-rs:RegionList[1]  XmpText type="Struct"' Stoneheng
$ exiv2 -M'set Xmp.mwg-rs.Regions/mwg-rs:RegionList[1]/mwg-rs:Area/stArea:x XmpText  0.6
$ exiv2 -M'set Xmp.mwg-rs.Regions/mwg-rs:RegionList[1]/mwg-rs:Area/stArea:t XmpText  0.3
$ exiv2 -M'set Xmp.mwg-rs.Regions/mwg-rs:RegionList[1]/mwg-rs:Area/stArea:h XmpText  0.4
$ exiv2 -M'set Xmp.mwg-rs.Regions/mwg-rs:RegionList[1]/mwg-rs:Area/stArea:w XmpText  0.3
$ exiv2 -pX Stonehenge.jpg | xmllint -pretty 1 -
<?xml version="1.0"?>
<?xpacket begin="" id="W5M0MpCehiHzreSzNTczkc9d"?>
<x:xmpmeta xmlns:x="adobe:ns:meta/" x:xmptk="XMP Core 4.4.0-Exiv2">
  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <rdf:Description xmlns:xmp="http://ns.adobe.com/xap/1.0/" xmlns:mwg-rs="..." xmlns:s
      <mwg-rs:Regions rdf:parseType="Resource">
        <mwg-rs:AppliedToDimensions stDim:w="912"/>
        <mwg-rs:RegionList>
          <rdf:Bag>
            <rdf:li rdf:parseType="Resource">
              <mwg-rs:Area stArea:x="0.6809" stArea:t="0.3537" stArea:h="0.4264" stArea
            </rdf:li>
          </rdf:Bag>
        </mwg-rs:RegionList>
      </mwg-rs:Regions>
    </rdf:Description>
  </rdf:RDF>
</x:xmpmeta>
<?xpacket end="w"?>```
```

Adobe XMPsdk

The Adobe XMPsdk is available here: https://github.com/adobe/XMP-Toolkit-SDK.git. In addition to the code, there is documentation, CMake build scripts and sample applications. The current documentation is:

```Bash
550 rmills@rmillsmbp:~/gnu/github $ ls -l ~/Google\ Drive/PDFs/XMP/
total 9536
-rw-r--r--@ 1 rmills  staff   311694 18 Sep 19:02 XMP-Toolkit-SDK-Overview.pdf
-rw-r--r--@ 1 rmills  staff   192908 18 Sep 14:27 XMPAddendumProgrammersGuide.pdf
-rw-r--r--@ 1 rmills  staff   244176 18 Sep 14:27 XMPFilesPluginSDK.pdf
-rw-r--r--@ 1 rmills  staff  2683121 18 Sep 14:27 XMPProgrammersGuide.pdf
-rw-r--r--@ 1 rmills  staff   383354 18 Sep 14:27 XMPSpecificationPart2.pdf
-rw-r--r--@ 1 rmills  staff  1366689 18 Sep 14:27 XMPSpecificationPart3.pdf
551 rmills@rmillsmbp:~/gnu/github $
```

Adobe XMPsdk creates two libraries XMPFiles and XMPCore. XMPFiles implements extracting/inserting XMP into various file many formats such as JPEG, PNG and TIFF. These are specified in XMPSpecificationPart3. From an Exiv2 point of view, the file handling library is of little interest as Exiv2 has its own file handlers.

The program tvisitor can extract XMP from image files. However it doesn't use XMPsdk for several reasons. Firstly, tvisitor knows how to navigate images and extract XMP without using the library XMPFiles. Secondly, I want tvisitor.cpp to be a "one file" application with no dependencies. Thirdly, I have not studied XMPCore and do not need to use it in tvisitor.cpp.

Building Adobe XMPsdk using Adobe's build environments

I have never built Adobe XMPsdk with Visual Studio, Cygwin, MinGW or UNIX. On macOS, I can build the libraries, but not the samples.

Here's how I have built XMPsdk on Ubuntu 20.04 with GCC 9.3.0

1. git clone https://github.com/adobe/XMP-TOOLKIT-SDK.git
2. git clone https://github.com/libexpat/libexpat.git
3. git clone https://github.com/madler/zlib.git
4. mkdir -p XMP-TOOLKIT-SDK/third-party/expat/lib ; cp -v zlib/*.? XMP-TOOLKIT-SDK/third-party/zlib/
5. mkdir -p XMP-TOOLKIT-SDK/third-party/expat/lib ; cp -v libexpat/expat/lib/*.? XMP-TOOLKIT-SDK/third-party/expat/lib
6. EDIT XMP-TOOLKIT-SDK/third-party/expat/lib/xmlparse.c and insert the line `#define XML_POOR_ENTROPY` at the top of the file
7. mkdir -p XMP-TOOLKIT-SDK/tools/cmake/bin ; ln -s $(which cmake) XMP-TOOLKIT-SDK/tools/cmake/bin/cmake
8. Apply the 2 fixes in https://github.com/adobe/XMP-Toolkit-SDK/issues/8
9. $ cd XML-TOOLKIT_SDK/build ; make DynamicRelease64
10. $ cd ../samples/build ; make DynamicRelease64

The following build artefacts are created on Linux.

```Bash
rmills@ubuntu:~/gnu/github/XMP-TOOLKIT-SDK$ find public/ -type f
public/libraries/i80386linux_x64/release/libXMPCore.so
public/libraries/i80386linux_x64/release/libXMPFiles.so
public/include/XMP.incl_cpp
public/include/XMP_Const.h
...
rmills@ubuntu:~/gnu/github/XMP-TOOLKIT-SDK$ find samples/target -type f
samples/target/i80386linux_x64/release/ModifyingXMPNewDOM
samples/target/i80386linux_x64/release/XMPIterations
samples/target/i80386linux_x64/release/XMPCoreCoverage
samples/target/i80386linux_x64/release/CustomSchema
samples/target/i80386linux_x64/release/XMPFilesCoverage
samples/target/i80386linux_x64/release/libXMPCore.so
samples/target/i80386linux_x64/release/DumpFile
samples/target/i80386linux_x64/release/DumpScannedXMP
samples/target/i80386linux_x64/release/CustomSchemaNewDOM
samples/target/i80386linux_x64/release/ReadingXMPNewDOM
samples/target/i80386linux_x64/release/ReadingXMP
samples/target/i80386linux_x64/release/libXMPFiles.so
samples/target/i80386linux_x64/release/ModifyingXMP
samples/target/i80386linux_x64/release/XMPCommand
samples/target/i80386linux_x64/release/DumpMainXMP
rmills@ubuntu:~/gnu/github/XMP-TOOLKIT-SDK$
```

If using macOS to build the libraries, follow steps 1..7 above then use build/GenerateXMPToolkitSDK_mac.sh to create the .xcodeproj files. You should be able to build the libraries with the following command:

```Bash
$ xcodebuild -project xcode/dynamic/intel_64_libcpp/XMPToolkitSDK64.xcodeproj -con
```

I was unable to get to work with Xcode 12.1 because it complained about the SDK and absense of Command line tools.

You can also build using the Xcode IDE by opening on of the generated projects such as xcode/dynamic/intel*64*libcpp/XMPToolkitSDK64.xcodeproj. The default build is “Debug” and you can change that to “Release” by editing the scheme which is presumably obvious to Xcode experts.

I had to manually download and install the Xcode command line tools for Xcode 12.1 and download and install /Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/MacOSX10.14.sdk

Although it build from the Xcode IDE, I never succeeded in getting to build with the xcodebuild command.

The following libraries are built:

```Bash
$ cd XMP-Toolkit-SDK/public/libraries/macintosh/intel_64_libcpp/Release
$ ls -l
drwxr-xr-x+ 5 rmills  staff       160 10 Nov 11:32 XMPCore.framework
drwxr-xr-x+ 5 rmills  staff       160 10 Nov 11:32 XMPFiles.framework
-rw-r--r--+ 1 rmills  staff  26254464 10 Nov 11:21 libXMPCoreStatic.a
-rw-r--r--+ 1 rmills  staff  34108944 10 Nov 11:21 libXMPFilesStatic.a
```

Using Adobe XMPsdk sample applications

There is a example in XMPProgrammersGuide.pdf page 73 of "Modifying XMP" in which the sample application operates on samples/testfiles/Image1.jpg

```
557 rmills@rmillsmbp:~/gnu/github/XMP-Toolkit-SDK $ exiv2 -px -g dc samples/testfi
Xmp.dc.format                                XmpText    10  image/jpeg
Xmp.dc.description                           LangAlt     1  lang="x-default" Wilting Ros
Xmp.dc.creator                               XmpSeq      1  XMP SDK
Xmp.dc.title                                 LangAlt     3  lang="x-default" An English
Xmp.dc.subject                               XmpBag      4  XMP, SDK, Test, File
558 rmills@rmillsmbp:~/gnu/github/XMP-Toolkit-SDK $
```

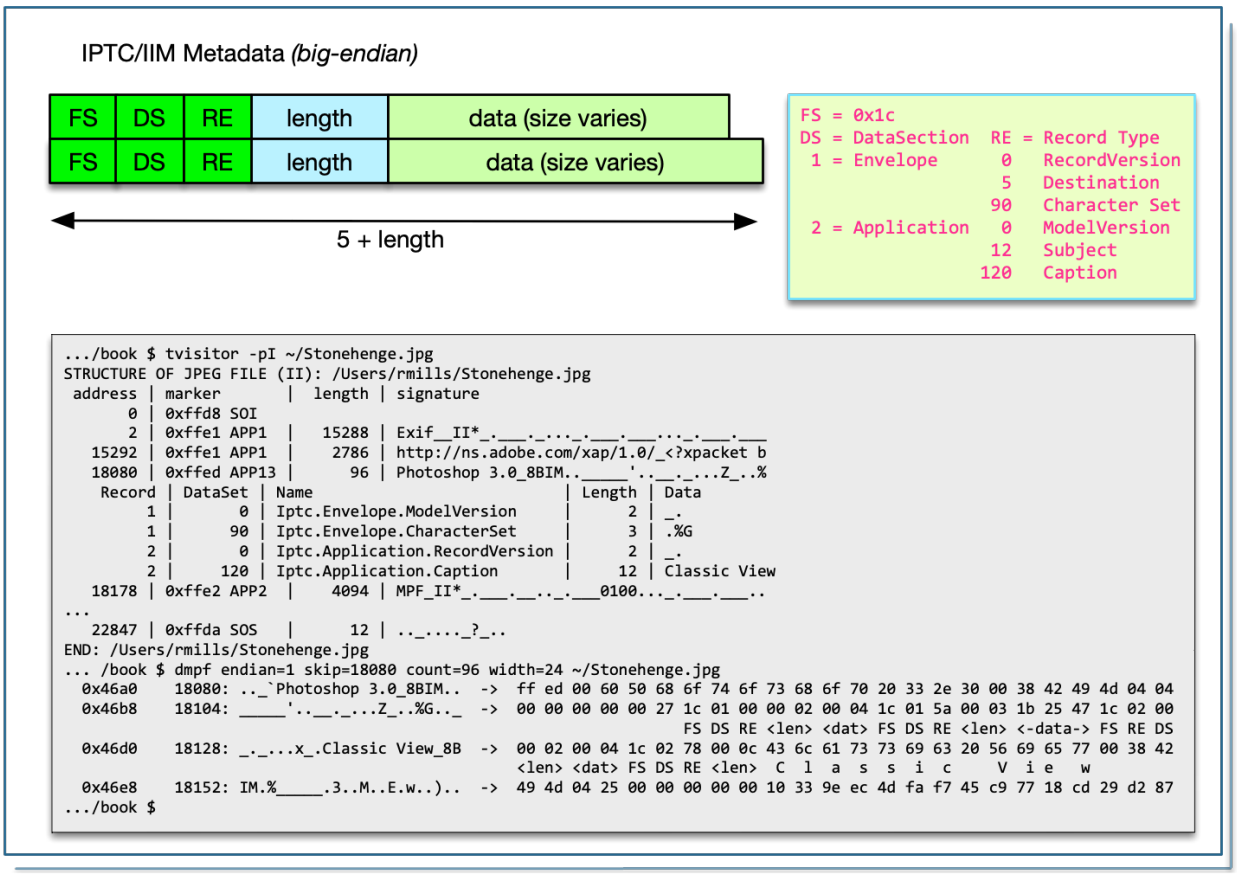
Bash

Building Adobe XMPsdk with Exiv2.

Exiv2 has a copy of XMPsdk included in the code base and builds easily. Exiv2 uses Conan to build and link other versions of XMPsdk. See README-CONAN.md

TOC

2.3 IPTC/IIM Metadata



This standard is championed by the International Press Telecommunications Council and predates both Exif and XMP.

The latest documentation (2014) is https://www.iptc.org/std/photometadata/specification/IPTC-PhotoMetadata-201407_1.pdf The implementation of IPTC in Exiv2 was added before I joined the project and I know very little about this matter. I'm pleased to say that the code is stable and reliable and I cannot recall any user raising an issue about IPTC.

As is common in standards, there are competing and overlapping standards for metadata that reflect the interests of their champions. So, Exif is for Cameras, XMP primarily for Application Programs, and IPTC is for the Press Industry. Being a software engineer, I know very little about how people actually use metadata. I belive IPTC preserves copyright and other high value resources as files move along the work-flow from the origin to a magazine or newspaper. There is another parallel trade association called The Metadata Working Group which works to define the use and meaning of metadata.
https://en.wikipedia.org/wiki/Metadata_Working_Group

There is a website that documents IPTC here: https://help.accusoft.com/ImageGear-Net/v22.1/Windows/HTML/topic371.html

Name	Section	Typical Values
Envelope	1	Destination, DateSent
Application	2	Subject, ObjectName
Digital News Photo	3	PictureNumber, ICCInputColourProfile
Pre-Object Descriptor	7	MaxSubfileSize
Object Record	8	SubFile
Post-Object Descriptor	9	ConfirmedObjectDataSize

I don't know why there are no sections 4, 5 or 6.

The Exiv2 support for IPTC is documented here: https://exiv2.org/iptc.html. I don't know why Exiv2 does not provide support for sections 3, 7, 8 or 9 as it could be easily added.

The code in tvisitor.cpp supports the following DataSets, all others are ignored.

Section	Record	Name
1. Envelope	0	RecordVersion
	5	Destination
	90	CharacterSet
2. Application	0	ModelVersion
	12	Subject
	120	Caption

There is considerably more information about DataSets in the Exiv2 code-base. I believe this defines the format of data values such as short and long. In the discussion about MakerNotes, I added code to decode binary data in tvisitor.cpp as this is a very important topic to understand in the Exiv2 code-base. I haven't studied the IPTC data to the same depth as I believe the tvisitor.cpp/IPTC support is sufficient to understand how IPTC data is stored and decoded.

```Bash
$ cp ~/Stonehenge.jpg .
$ exiv2 -M'set Iptc.Envelope.Destination Camberley Print Room' Stonehenge.jpg
$ exiv2 -M"set Iptc.Application2.Subject Robin's Book" Stonehenge.jpg
$ exiv2 -pi Stonehenge.jpg
Iptc.Envelope.ModelVersion                    Short       1  4
Iptc.Envelope.CharacterSet                    String      3  G
Iptc.Envelope.Destination                     String     20  Camberley Print Room
Iptc.Application2.RecordVersion               Short       1  4
Iptc.Application2.Caption                     String     12  Classic View
Iptc.Application2.Subject                     String     12  Robin's Book
```

The IPTC data in a JPEG is stored in the APP13 PhotoShop segment, as we see here:

```Bash
.../book $ tvisitor -pI ~/Stonehenge.jpg
STRUCTURE OF JPEG FILE (II): /Users/rmills/Stonehenge.jpg
 address | marker       |  length | signature
       0 | 0xffd8 SOI
       2 | 0xffe1 APP1  |   15288 | Exif__II*_.___._...._.___.___...._.___.___
   15292 | 0xffe1 APP1  |    2786 | http://ns.adobe.com/xap/1.0/_<?xpacket b
   18080 | 0xffed APP13 |      96 | Photoshop 3.0_8BIM.._____'.._._.._...Z_..%
    Record | DataSet | Name                          | Length | Data
         1 |       0 | Iptc.Envelope.ModelVersion    |      2 | _.
         1 |      90 | Iptc.Envelope.CharacterSet    |      3 | .%G
         2 |       0 | Iptc.Application.RecordVersion |      2 | _.
         2 |     120 | Iptc.Application.Caption      |     12 | Classic View
   18178 | 0xffe2 APP2  |    4094 | MPF_II*_.___.__.._._.___0100..._.___.___..
   22274 | 0xffdb DQT   |     132 | _.....................................
   22408 | 0xffc0 SOF0  |      17 | ....p..!_........
   22427 | 0xffc4 DHT   |     418 | __........_________............._.........
   22847 | 0xffda SOS   |      12 | .._...._?_..
END: /Users/rmills/Stonehenge.jpg
```

IPTC Character Set Encoding

CharacterSet is in the Envelope DataSection. CharacterSet is used by transmission protocols to transmit resources via modems and other resources that were in common use when IPTC was first defined in the 1990s. This field is set by the exiv2 convertors to "<esc>%G" to represent UTF-8. "<esc>" is ascii 0x01b (27)

I believe the data is defined in the Standard ISO/IEC 2022. The following web page has a section ***Interaction with other coding systems*** in which I discovered the following table.
https://en.wikipedia.org/wiki/ISO%2FIEC_2022

Unicode Format	Code(s)	Hex[84]
UTF-1	(UTF-1 not in current ISO/IEC 10646.)	
UTF-8	ESC % G , ESC % / I	1B 25 47 ,[86] 1B 25 2F 49 [87]
UTF-16	ESC % / L	1B 25 2F 4C [88]
UTF-32	ESC % / F	1B 25 2F 46

IPTC Extended Blocks

As with Exif metdata, the IPTC data block can exceed 64k byte and this cannot be stored in a single JPEG segment. Exiv2 has code to deal with this and is documented here: https://dev.exiv2.org/issues/0000533

IPTC in Tiff and other formats.

In Tiff, IPTC data is contained in the following tag:

```Bash
$ taglist ALL | grep Image.IPTCNAA | head -1 | csv -
[Image.IPTCNAA] [33723] [0x83bb]    [Image] [Exif.Image.IPTCNAA]    [Long]  [Contains ar
$
```

In PNG files, the signature **Raw profile type iptc_* is used.

```Bash
$ exiv2 -pS test/data/ReaganLargePng.png
STRUCTURE OF PNG FILE: test/data/ReaganLargePng.png
 address | chunk |  length | data                              | checksum
       8 | IHDR  |      13 | ............                      | 0x8cf910c3
      33 | zTXt  |    8461 | Raw profile type exif..x...iv.    | 0x91fbf6a0
    8506 | zTXt  |     636 | Raw profile type iptc..x..TKn.    | 0x4e5178d3
    9154 | iTXt  |    7156 | XML:com.adobe.xmp.....<?xpacke    | 0x8d6d70ba
   16322 | gAMA  |       4 | ....                              | 0x0bfc6105
   16338 | iCCP  | 1151535 | ICC profile..x...UP.........!!    | 0x11f49e31
 1167885 | bKGD  |       6 | ......                            | 0xa0bda793
 1167903 | pHYs  |       9 | ...#...#.                         | 0x78a53f76
 1167924 | tIME  |       7 | ......2                           | 0x582d32e4
 1167943 | zTXt  |     278 | Comment..x.}..n.@....0..5..h..    | 0xdb1dfff5
...

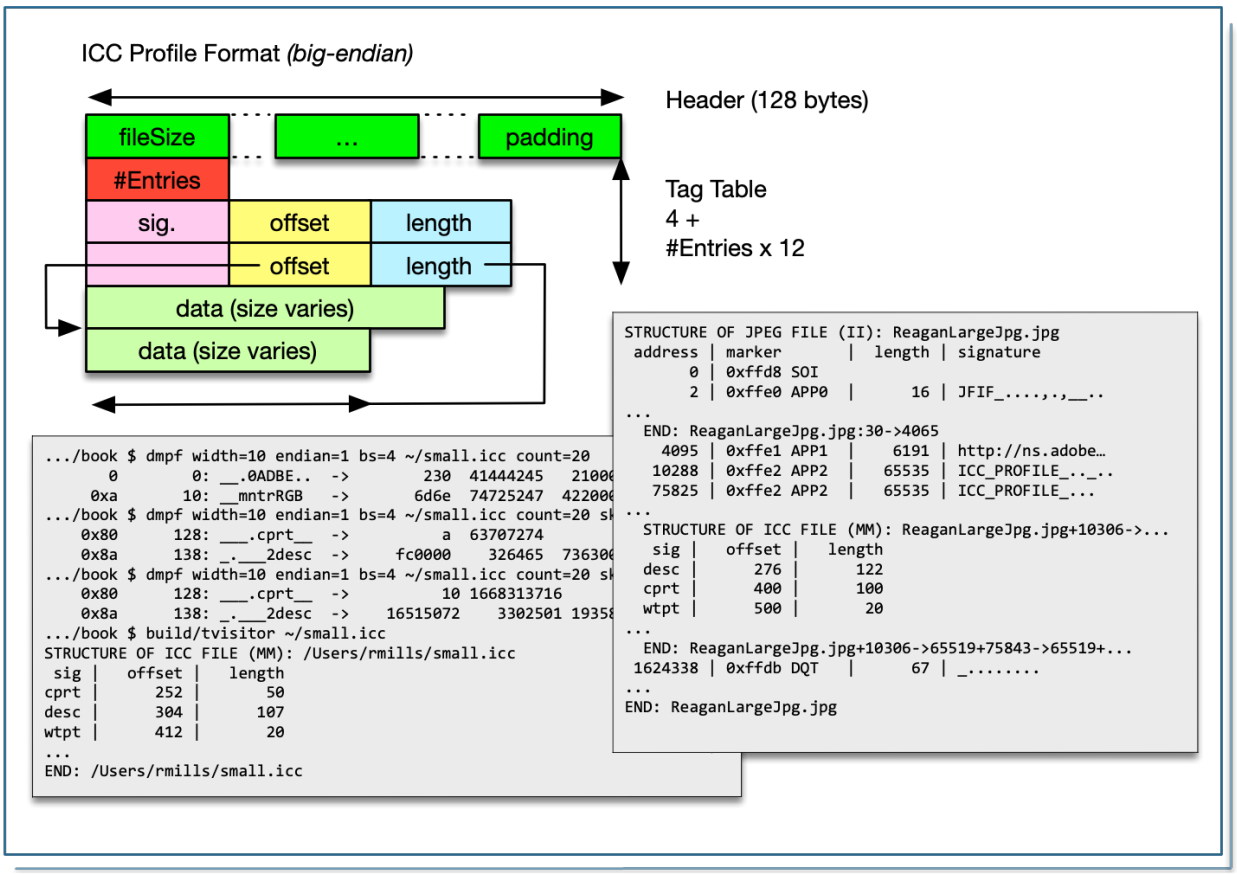
In JP2 file, IPTC is embedded in the uuid/iptc box.

```bash
$ tvisitor -pR ~/jp2.jp2
STRUCTURE OF JP2 FILE (MM): /Users/rmills/jp2.jp2
 address |   length | box             | uuid | data
       0 |        4 | 0x2020506a jP   |      | ....
      12 |       12 | 0x70797466 ftyp |      | jp2 ____jp2
      32 |       37 | 0x6832706a jp2h |      | ___.ihdr___.___._...._____.colr._____.
  STRUCTURE OF JP2 FILE (MM): /Users/rmills/jp2.jp2:40->37
         0 |       14 | 0x72646869 ihdr |      | ___.___._...__
        22 |        7 | 0x726c6f63 colr |      | ._____.
  END: /Users/rmills/jp2.jp2:40->37
      77 |     1334 | 0x64697575 uuid | exif | II*_.___._..._..__.___..._.___>.__..._._
    1419 |      934 | 0x64697575 uuid | iptc |             040621-N-6536T-062     5
```
```

```
31     2361 |     5582 | 0x64697575 uuid |  xmp | <?xpacket begin="..." id="W5M0MpCehiHzre
32     7951 |    32650 | 0x6332706a jp2c |      | .O.Q_/_____.___.__________.___.________
33 END: /Users/rmills/jp2.jp2
```

TOC

2.4 ICC Profile



The ICC Profile is a standand alone file that can be embedded verbatim in many image formats. The purpose of the ICC profile is to provide additional color data about the image. Most colour images are encoded as RGB or CMYK. When these are rendered on a device, it's necessary to know the actual colour of Red in the image and on the output device. The Colour Management System (CMS) attempts to render the image to be the same on different devices. This is of course impossible, however the aim of the ICC Profile is enable the software to achieve good colour fidelity when printing or displaying on different devices.

The ICC Profile is a member of the TIFF family of image standards. It has a header, a directory of "tags" and values for the tags.

Exiv2 has no code to inspect or modify the contents of the ICC Profile. The data is treated as a binary "blob". You can insert/delete/add/replace the ICC Colour Profile in several image formats including JPEG, JP2, PNG and TIFF.

The code which accompanies this book can inspect the contents of an ICC profile.

The specification is available here: http://www.color.org/icc_specs2.xalter. I believe the current ICC Profile Specification is: ICC.2-2016-7.pdf

TOC

2.5 MakerNotes

https://exiv2.org/makernote.html

MakerNotes are usually written as an IFD, however most manufacturers have extra bytes that precede the IFD. I suspect the extra bytes are version information. The code in tvisitor.cpp to handle the makernotes is:

```C++
void IFD::visitMakerNote(Visitor& visitor,DataBuf& buf,uint16_t count,uint32_t offs
{
    if ( image_.maker_ == kNikon ) {
        // Nikon MakerNote is embeded tiff `II*_....` 10 bytes into the data!
        size_t punt = buf.strequals("Nikon") ? 10
                    : 0
                    ;
        Io     io(io_,offset+punt,count-punt);
        TiffImage makerNote(io,image_.maker_);
        makerNote.visit(visitor,makerDict());
    } else if ( image_.maker_ == kAgfa && buf.strequals("ABC") ) {
        // Agfa  MakerNote is an IFD `ABC_II#E...`  6 bytes into the data!
        ImageEndianSaver save(image_,keLittle);
        IFD makerNote(image_,offset+6,false);
        makerNote.visit(visitor,makerDict());
    } else {
        bool   bNext = maker()  != kSony;                                    // Sony
        size_t punt  = maker()  == kSony && buf.strequals("SONY DSC ") ? 12 : 0; // Sony
        IFD makerNote(image_,offset+punt,bNext);
        makerNote.visit(visitor,makerDict());
    }
} // visitMakerNote
```

To be written.

TOC

2.6 Metadata Convertors

Exiv2 has code to convert data between different Metdata standards. Generally when you update Exif metadata, equivalent modifications will be performed on the IPTC and XMP metadata. I can't explain why this code was added to Exiv2 and, while it may be convenient and invisible in its operation, it also has undesirable side effects.

If Exiv2 is ever rewritten, the decision to keep this capability should be carefully reviewed. I think it would be better to not have this at all and leave library users to provide this in their application code.

TOC

3 Reading Metadata

3.1 Read metadata with dd

The exiv2 option ***-pS*** prints the structure of an image.

```Bash
$ exiv2 -pS ~/Stonehenge.jpg
STRUCTURE OF JPEG FILE: /Users/rmills/Stonehenge.jpg
 address | marker       |  length | data
       0 | 0xffd8 SOI
       2 | 0xffe1 APP1  |   15288 | Exif..II*.....................
   15292 | 0xffe1 APP1  |    2610 | http://ns.adobe.com/xap/1.0/.<?x
   17904 | 0xffed APP13 |      96 | Photoshop 3.0.8BIM.......'.....
   18002 | 0xffe2 APP2  |    4094 | MPF.II*..............0100.....
   22098 | 0xffdb DQT   |     132
   22232 | 0xffc0 SOF0  |      17
   22251 | 0xffc4 DHT   |     418
   22671 | 0xffda SOS
$
```

We can see that the Exif metadata is stored at offset=2+2+2+6 and has length 15288-offset. We can extract that as follows:

```Bash
$ dd if=~/Stonehenge.jpg count=$((15288-(2+2+2+6))) bs=1 skip=$((2+2+2+6)) > foo.t
15276+0 records in
15276+0 records out
15276 bytes transferred in 0.102577 secs (148922 bytes/sec)
$ dd if=~/Stonehenge.jpg count=$((15288-(2+2+2+6))) bs=1 skip=$((2+2+2+6)) | dmpf - | he
       0        0: II*_.___._...._.___.___..._.___._  ->  49 49 2a 00 08 00 ...
.../book $
$ file foo.tif
foo.tif: TIFF image data, little-endian, direntries=11, manufacturer=NIKON CORPORATION,
$ exiv2 -pa foo.tif
Warning: Directory Thumbnail, entry 0x0201: Data area exceeds data buffer, ignoring it.
Exif.Image.Make                              Ascii      18  NIKON CORPORATION
Exif.Image.Model                             Ascii      12  NIKON D5300
Exif.Image.Orientation                       Short       1  top, left
Exif.Image.XResolution                       Rational    1  300
Exif.Image.YResolution                       Rational    1  300
...
```

Internally, this is exactly how exiv2 works. It doesn't use ***dd*** of course. However it identifies the Exif IFD and parses it into memory.

Using dd is a useful trick to recover data which can be easily seen in the file. For example, if you wished to extract the pixels from an image, dd can extract them. Of course you have to determine the offset and length to extract and exiv2 has excellent tools to provide that data.

You can extract and inspect the metadata with this single *rather elegant* command:

```
$ dd if=~/Stonehenge.jpg count=$((15288-(2+2+2+6))) bs=1 skip=$((2+2+2+6)) 2>/dev/
Exif.Image.Make                              Ascii      18  NIKON CORPORATION
Exif.Image.Model                             Ascii      12  NIKON D5300
Exif.Image.Orientation                       Short       1  top, left
$
```

Bash

The exiv2 command ***exiv2 -pS image*** reveals the structure of a file with | separated fields. The data is presented to look nice. However it's also very convenient for parsing in bash with the utility ***cut***:

```
$ image=~/Stonehenge.jpg
$ exiv2 -pS $image 2>/dev/null | grep APP1 | grep Exif
$        2 | 0xffe1 APP1  |   15288 | Exif..II*.....................
$ line=$(exiv2 -pS ~/Stonehenge.jpg 2>/dev/null | grep APP1 | grep Exif )
$ start=$(echo $line|cut  -d'|' -f 1)
$ count=$(echo $line|cut  -d'|' -f 3)
$ dd if=$image count=$((count-10)) bs=1 skip=$((start+10)) 2>/dev/null | exiv2 -pa - 2>/
Exif.Image.Make                              Ascii      18  NIKON CORPORATION
Exif.Image.Model                             Ascii      12  NIKON D5300
Exif.Image.Orientation                       Short       1  top, left
$
```

Bash

You may be interested to discover that option ***-pS*** which arrived with Exiv2 v0.25 was joined in Exiv2 v0.26 by ***-pR***. This is a ***recursive*** version of ***-pS***. It dumps the structure not only of the file, but also subfiles (such as IFDs and JPEG/thumbnails and ICC profiles). This is discussed in detail here: 3.4 IFD::accept().

TOC

3.2 Tags and TagNames

A tag is the unit of data storage in a tiff entry. It has a tag (uint16_t), type (uint16_t), count (uint32_t) and value (uint32_t)_ The meaning of the 16-bit tag is defined by the standard to which the IFD has been written. These are discussed here: 2.1 Exif Metadata. The TIFF-EP specification defines tags in the IFD0 (the "top-level" IFD) such as Make (0x010f) and ExifTag (0x8769). The tag ExifTag introduces a new IFD in which tags of interest to the Exif Committee are defined. Examples are DateTimeOriginal (0x9003) and MakerNote (0x927c), GpsTag (ktGps). Tags such as MakerNote and GpsTag define new IFDs. The meaning for the tags in the IFD referenced by GpsTag is defined by the Exif Committee. The meaning of the tags in the IFD referenced by a MakerNote have been discovered by reverse engineering.

It's important to appreciate that when you visit an IFD, you need a dictionary of tag->name to know the meaning of the tag. That dictionary is not a constant, it depends on the IFD that is being read. In the case of the MakerNote, the dictionary of tag->name depends on the Manufacturer. The tvisitor.cpp program invokes code to set the makerDict when it reads the Make (0x010f) in the "top-level" IFD.

Exiv2 (and tvisitor.cpp) report tags with the syntax such as Exif.Image.Make. Exif.Photo.DateTimeOriginal. This syntax is of the format: Family.Group.Tagname. There are three Families in Exiv2 which are Exif, IPTC and Xmp. The group Image implies that the tag was read in IFD0, the group Photo implies that the tag was read in the the Exif IFD. tvisitor.cpp has about 10 groups (Image, Photo, GPS, Nikon, Apple, Canon etc). Exiv2 has 106 groups as each of about 10 manufacturers have about 10 sub groups.

For simplicity, tvisitor.cpp only supports the family Exif, however it has code to decode and present IPTC, ICC and Xmp metadata.

TOC

3.3 Visitor Design Pattern

This is implemented using ***Visitor*** in Design Patterns)

The concept in the visitor pattern is to separate the data in an object from the code which that has an interest in the object. In the following code, we have a vector of students and every student has a name and an age. We have several visitors. The French Visitor translates the names of the students. The AverageAgeVisitor calculates the average age of the visitor. Two points to recognise in the pattern:

1. The students know nothing about the visitors. However, they know when they are visited. If the visitor has an API, the students can obtain data about the visitor.

2. The visitors use the student API to get data about a student.

C++

```
// visitor.cpp
#include <iostream>
#include <string>
#include <vector>
#include <map>

// 1.  declare types
class   Student; // forward

// 2. Create abstract "visitor" base class with an element visit() method
class Visitor
{
public:
    Visitor() {};
    virtual void visit(Student& student) = 0 ;
};

// 3. Student has an accept(Visitor&) method
class Student
{
public:
    Student(std::string name,int age,std::string course)
    : name_(name)
    , age_(age)
    , course_(course)
    {}
    void accept(class Visitor& v) {
      v.visit(*this);
    }
    std::string name()  { return name_; }
    int         age()   { return age_;  }
    std::string course(){ return course_;  }
private:
    std::string course_ ;
    std::string name_   ;
```

```
36      int          age_     ;
37  };
38
39  // 4. Create concrete "visitors"
40  class RollcallVisitor: public Visitor
41  {
42  public:
43      RollcallVisitor() {}
44      void visit(Student& student)
45      {
46          std::cout << student.name() <<  " | " << student.age() << " | " << student.cours
47      }
48  };
49
50  class FrenchVisitor: public Visitor
51  {
52  public:
53      FrenchVisitor()
54      {
55          dictionary_["this"]       = "ce"       ;
56          dictionary_["that"]       = "que"      ;
57          dictionary_["the other"] = "l'autre" ;
58      }
59      void visit(Student& student)
60      {
61          std::cout << "FrenchVisitor: " << dictionary_[student.name()] << std::endl;
62      }
63  private:
64      std::map<std::string,std::string> dictionary_;
65  };
66
67  class AverageAgeVisitor: public Visitor
68  {
69  public:
70      AverageAgeVisitor() : students_(0), years_(0) {}
71      void visit(Student& student)
72      {
73          students_ ++ ;
74          years_    += student.age();
75      }
76      void reportAverageAge()
77      {
78          std::cout << "average age = "  << (double) years_ / (double) students_ << std::e
79      }
80  private:
81      int years_;
82      int students_;
83  };
```

And let's create a container for Students.

```C++
class College
{
public:
             College() {};
    virtual ~College() {};

    void add(Student student) {
        students_.push_back(student);
    }
    void visit(Visitor& visitor) {
        for ( std::vector<Student>::iterator student = students_.begin() ; student != st
            student->accept(visitor);
        }
    }
private:
    std::vector<Student> students_;
};
```

Create an application with data.

```C++
int main() {
    // create a highSchool and add some students
    College highSchool;

    highSchool.add(Student("this",10,"art"             ));
    highSchool.add(Student("that",12,"music"           ));
    highSchool.add(Student("the other",14,"engineering"));

    // Create different visitors to visit highSchool
    RollcallVisitor  rollCaller;
    highSchool.visit(rollCaller);

    FrenchVisitor    frenchVisitor;
    highSchool.visit(frenchVisitor);

    AverageAgeVisitor averageAgeVisitor;
    highSchool.visit(averageAgeVisitor);
    averageAgeVisitor.reportAverageAge();

    return 0 ;
}
```

And when we run it:

```Bash
.../book/build $ ./visitor
this | 10 | art
that | 12 | music
the other | 14 | engineering
FrenchVisitor: ce
FrenchVisitor: que
FrenchVisitor: l'autre
average age = 12
.../book/build $
```

We could of course add other classes to this program. We could have **class Building** and add buildings to the

college. The visitor could visit all the buildings. We could have rooms in every building. I am sure you get the idea.

In a JPEG, we have a linked list of segments. So tvisitor.cpp has a visitSegment() method. As JPEG has an embedded Exif Tiff, so we have visitExif(), visitIFD(), visitTag(), visitXMP(). The visitor knows nothing about how to navigate the file.

In tvisitor.cpp, we only have a single Visitor called ReportVisitor. When you create him, you specify options which are Basic, Recursive, XMP. The ReportVisitor effectively performs the same options as ***$ exiv2 -pS***, or ***$ exiv2 -pR***, or ***$ exiv2 -pX***. We could easily create a new class Exiv2Visitor which would create Exiv2::ExifData. It's also possible to create a class Exiv2Writer which would output a new file with modified metadata.

TOC

3.4 IFD::accept() and TiffImage::accept()

Exiv2 has two tiff parsers - TiffVisitor and Image::printIFDStructure(). TiffVisitor was written by Andreas Huggel. It's very robust and has been almost bug free for 15 years. I wrote the parser in Image::printIFDStructure() to try to understand the structure of a tiff file. The code in Image::printIFDStructure() is easier to understand.

The code which accompanies this book has a simplified version of Image::printIFDStructure() called IFD::accept() and that's what will be discussed here. The code that accompanies this book is explained here: Code discussed in this book

It is important to realise that metadata is defined recursively. In a Tiff File, there will be a Tiff Record containing the Exif data (written in Tiff Format). Within, that record, there will be a MakerNote which is usually written in TIFF Format. TIFF Format is referred to as an IFD - an Image File Directory.

TiffImage::accept() uses a simple direct approach to parsing the tiff file. When another IFD is located, IFD::accept() is called recursively. As a TIFF file has an 8 byte header which provides the offset to the first IFD. We can descend into the tiff file from the beginning. For other files types, the file handler has to find the Exif IFD and then call IFD::accept().

There are several ways in which IFD::accept() is called. TiffImage::accept() starts with the tiff header ***II*_long*** or ***MM_*long*** and then calls ***IFD::accept()***. Makernotes are usually an IFD. Some manufactures (Nikon) embed a Tiff. Some (Canon and Sony) embed an IFD. It's quite common (Sony) to embed a single IFD which is not terminated with a four byte null uint32_t.

The program tvisitor has several file handlers such as TiffImage, JpegImage and CrwImage. Exiv2 has handlers for about 20 different formats. If you understand Tiff and Jpeg, the others are boring variations.

```C++
void IFD::accept(Visitor& visitor,const TagDict& tagDict/*=tiffDict*/)
{
    IoSave   save(io_,start_);
    bool     bigtiff = image_.bigtiff();
    endian_e endian  = image_.endian();

    if ( !image_.depth_++ ) image_.visits().clear();
    visitor.visitBegin(image_);
    if ( image_.depth_ > 100 ) Error(kerCorruptedMetadata) ; // weird file
```

```
10
11      // buffer
12      DataBuf  entry(bigtiff ? 20 : 12);
13      uint64_t start=start_;
14      while  ( start ) {
15          // Read top of directory
16          io_.seek(start);
17          io_.read(entry.pData_, bigtiff ? 8 : 2);
18          uint64_t nEntries = bigtiff ? getLong8(entry,0,endian) : getShort(entry,0,endiar
19
20          if ( nEntries > 500 ) Error(kerTiffDirectoryTooLarge,nEntries);
21          visitor.visitDirBegin(image_,nEntries);
22          uint64_t a0 = start + (bigtiff?8:2) + nEntries * entry.size_; // addresss to rec
23
24          // Run along the directory
25          for ( uint64_t i = 0 ; i < nEntries ; i ++ ) {
26              const uint64_t address = start + (bigtiff?8:2) + i* entry.size_ ;
27              if ( visits().find(address) != visits().end()  ) { // never visit the same p
28                  Error(kerCorruptedMetadata);
29              }
30              visits().insert(address);
31              io_.seek(address);
32
33              io_.read(entry);
34              uint16_t tag    = getShort(entry,  0,endian);
35              type_e   type   = getType (entry,  2,endian);
36              uint64_t count  = get4or8 (entry,4,0,endian);
37              uint64_t offset = get4or8 (entry,4,1,endian);
38
39              if ( !typeValid(type,bigtiff) ) {
40                  Error(kerInvalidTypeValue,type);
41              }
42
43              uint64_t size   = typeSize(type) ;
44              size_t   alloc  = size*count     ;
45              DataBuf  buf(alloc);
46              if ( alloc < (bigtiff?8:4) ) {
47                  buf.copy(&offset,size);
48              } else {
49                  IoSave save(io_,offset);
50                  io_.read(buf);
51              }
52              if ( tagDict == tiffDict && tag == ktMake ) image_.setMaker(buf);
53              visitor.visitTag(io_,image_,address,tag,type,count,offset,buf,tagDict);  //
54
55              // recursion anybody?
56              if ( isTypeIFD(type) ) tag  = ktSubIFD;
57              switch ( tag ) {
58                  case ktGps       : IFD(image_,offset,false).accept(visitor,gpsDict );bre
59                  case ktExif      : IFD(image_,offset,false).accept(visitor,exifDict);bre
60                  case ktMakerNote :         visitMakerNote(visitor,buf,count,offset);brec
61                  case ktSubIFD    :
62                       for ( uint64_t i = 0 ; i < count ; i++ ) {
63                           offset = get4or8 (buf,0,i,endian);
64                           IFD(image_,offset,false).accept(visitor,tagDict);
65                       }
```

```
66                  break;
67                  default: /* do nothing */ ; break;
68              }
69          } // for i < nEntries
70
71          start = 0; // !stop
72          if ( next_ ) {
73              io_.seek(a0);
74              io_.read(entry.pData_, bigtiff?8:4);
75              start = bigtiff?getLong8(entry,0,endian):getLong(entry,0,endian);
76          }
77          visitor.visitDirEnd(image_,start);
78      } // while start != 0
79
80      visitor.visitEnd(image_);
81      image_.depth_--;
82  } // IFD::accept
```

The MakerNote is thorny. Every manufacturer has similar ideas with different details. This is discussed in detail: 2.5 MakerNotes

To complete the story, here's TiffImage::valid() and TiffImage::accept(). We need two flavours of accept. The default assumes tiffDict. The makernote handlers pass their TagDict to accept().

```C++
bool TiffImage::valid()
{
    IoSave restore(io(),0);

    // read header
    DataBuf  header(16);
    io_.read(header);

    char c    = (char) header.pData_[0] ;
    char C    = (char) header.pData_[1] ;
    endian_  = c == 'M' ? keBig : keLittle;
    magic_   = getShort(header,2,endian_);
    bigtiff_ = magic_ == 43;
    start_   = bigtiff_ ? getLong8(header,8,endian_) : getLong (header,4,endian_);
    format_  = bigtiff_ ? "BIGTIFF"                  : "TIFF"                    ;

    uint16_t bytesize = bigtiff_ ? getShort(header,4,endian_) : 8;
    uint16_t version  = bigtiff_ ? getShort(header,6,endian_) : 0;

    return (magic_ == 42||magic_ == 43) && (c == C) && ( c == 'I' || c == 'M' ) && bytes
} // TiffImage::valid

void TiffImage::accept(class Visitor& visitor)
{
    accept(visitor,tiffDict);
}

void TiffImage::accept(Visitor& visitor,TagDict& tagDict)
{
    if ( valid() ) {
        IFD ifd(*this,start_,next_);
        ifd.visit(visitor,tagDict);
    } else {
        std::ostringstream os ; os << "expected " << format_ ;
        Error(kerInvalidFileFormat,io().path(), os.str());
    }
} // TiffImage::visit
```

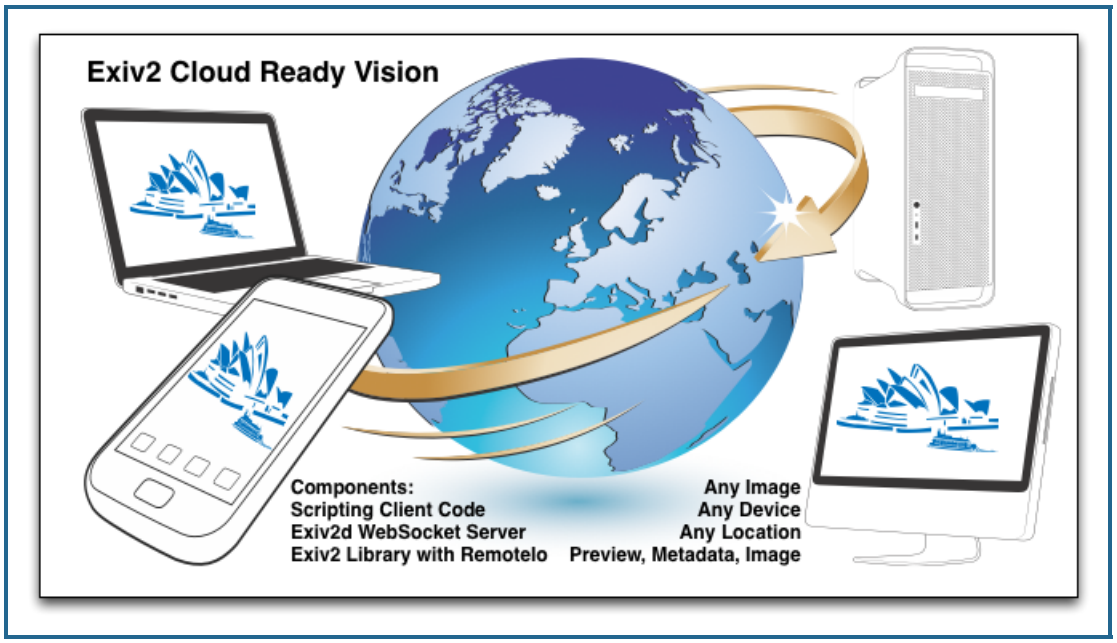
JpegImage::accept() navigates the chain of segments. It is discussed in detail: 8.8 Jpeg::Image accept()

When JpegImage::accept() finds the embedded TIFF in the APP1 segment, he does this. This is very similar to how the TiffImage for the Nikon makernote is created and navigated.

```C++
void ReportVisitor::visitExif(Io& io)
{
    if ( option() & kpsRecursive ) {
        // Beautiful.  io is a tiff file, call TiffImage::accept(visitor)
        TiffImage(io).accept(*this);
    }
}
```

He creates a TiffImage with the stream and calls TiffImage::accept(visitor). Software seldom gets simpler, as beautiful, or more elegant than this.

Just to remind you, BasicIo supports http/ssh and other protocols. This code will recursively descend into a remote file without copying it locally. And he does it with great efficiency. This is discussed in section 5 I/O in Exiv2



The code in ***tvisitor.cpp*** implements the visitor pattern and three visitors are implemented.

tvisitor option	*exiv2 option*	**Description**
$ ./tvisitor S path $ ./tvisitor path	$ exiv2 -pS path	Print the structure of the image
$ ./tvisitor R path	$ exiv2 -pR path	Recursively print the structure of the image
$ ./tvisitor X path	$ exiv2 -pX path	Print the XMP/xml in the image
$ ./tvisitor C path	$ exiv2 -pC path	Print the ICC Profile in the image
$ ./tvisitor I path	$ exiv2 -pi path	Print IPTC data
$ ./tvisitor U path	$ exiv2 -pa –undefined path	Show unknown tags

Let's see the recursive version in action:

```Bash
$ ./tvisitor R ~/Stonehenge.jpg
STRUCTURE OF JPEG FILE: /Users/rmills/Stonehenge.jpg
```

```
 address | marker       |  length | data
       0 | 0xffd8 SOI
       2 | 0xffe1 APP1  |   15288 | Exif__II*_.___._...._.___.___..._._
  STRUCTURE OF TIFF FILE (II): /Users/rmills/Stonehenge.jpg:12->15280
   address |    tag                             |      type |    count |    offset | v
        10 | 0x010f Make                        |     ASCII |       18 |       146 | N
        22 | 0x0110 Model                       |     ASCII |       12 |       164 | N
...
       118 | 0x8769 ExifTag                     |      LONG |        1 |           | 2
    STRUCTURE OF TIFF FILE (II): /Users/rmills/Stonehenge.jpg:12->15280
     address |    tag                             |      type |    count |    offset |
         224 | 0x829a ExposureTime                |  RATIONAL |        1 |       732 |
         236 | 0x829d FNumber                     |  RATIONAL |        1 |       740 |
...
         416 | 0x927c MakerNote                   | UNDEFINED |     3152 |       914 |
      STRUCTURE OF TIFF FILE (II): /Users/rmills/Stonehenge.jpg:12->15280:924->3142
       address |    tag                             |      type |    count |    offset
            10 | 0x0001 Version                     | UNDEFINED |        4 |
...
    END /Users/rmills/Stonehenge.jpg:12->15280
       130 | 0x8825 tag 34853 (0x8825)          |      LONG |        1 |           | 4
...
      4410 | 0x0213 YCbCrPositioning            |     SHORT |        1 |           | 1
  END /Users/rmills/Stonehenge.jpg:12->15280
   15292 | 0xffe1 APP1  |    2610 | http://ns.adobe.com/xap/1.0/_<?xpa
   17904 | 0xffed APP13 |      96 | Photoshop 3.0_8BIM.._____'..__._..
   18002 | 0xffe2 APP2  |    4094 | MPF_II*_.___.__..._.___0100..._.___
   22098 | 0xffdb DQT   |     132
   22232 | 0xffc0 SOF0  |      17
   22251 | 0xffc4 DHT   |     418
   22671 | 0xffda SOS
```

You can see that he identifies the file as follows:

```
         416 | 0x927c MakerNote                   | UNDEFINED |     3152 |
      STRUCTURE OF TIFF FILE (II): /Users/rmills/Stonehenge.jpg:12->15280:924->3142
       address |    tag                             |      type |    count |    offset
            10 | 0x0001 Version                     | UNDEFINED |        4 |
...
```

Bash

He is working on an embedded TIFF which is located at bytes 12..15289 which is the Tiff IFD. While processing that, he encountered a MakerNote which occupies bytes 924..3142 of that IFD. As you can see, its four bytes **0211**. You could locate that data with the command:

```
$ dd if=~/Stonehenge.jpg bs=1 skip=$((12+924+10+8)) count=4 2>/dev/null ; echo
0211
$
```

Bash

Using dd to extract metadata is discussed in more detail here: 8.1 Read metadata with dd.

Please be aware that there are two ways in which IFDs can occur in the file. They can be an embedded TIFF which is complete with the **II*_long** or **MM_*long** 8-byte header and the offset leads to the IFD. Or the IFD can be in the file without the header. IFD::visit(visitor) knows that the tags such as GpsTag and ExifTag are

IFDs and recursively calls IFD::visit(visitor). For the embedded TIFF (such as Nikon MakerNote), IFD::visit(visitor) creates a TiffImage and calls TiffImages.accept(visitor) which validates the header and calls IFD::visit(visitor).

Another important detail is that although the Tiff Specification expects the IFD to end with a uint32_t offset == 0, Sony (and other) maker notes do not. The IFD begins with a uint32_t to define length, followed by 12 byte tags. There is no trailing null uint32_t.

TOC

3.5 ReportVisitor::visitTag()

I added support in tvisitor.cpp for one binary tag which is Nikon Picture Control tag = 0x0023. You'll see from the output of tvisitor that it's 58 bytes.

```Bash
.../book/build $ ./tvisitor -pR ~/Stonehenge.jpg | grep -i picture
   286 | 0x0023 Exif.Nikon.PictureControl  | UNDEFINED |   58 |     | 0100STANDARD_______
.../book/build $
```

Beautifully documented as follows:

ExifTool	Exiv2
https://exiftool.org/TagNames/Nikon.html#PictureControl	https://exiv2.org/tags-nikon.html

Nikon PictureControl Tags

Index1	Tag Name	Writable	Values / Notes
0	PictureControlVersion	no	
4	PictureControlName	string[20]	
24	PictureControlBase	string[20]	
48	PictureControlAdjust	int8u	0 = Default Settings 1 = Quick Adjust 2 = Full Control
49	PictureControlQuickAdjust	int8u	
50	Sharpness	int8u	
51	Contrast	int8u	
52	Brightness	int8u	
53	Saturation	int8u	
54	HueAdjustment	int8u	
55	FilterEffect	int8u	0x80 = Off 0x81 = Yellow 0x82 = Orange 0x83 = Red 0x84 = Green 0xff = n/a
56	ToningEffect	int8u	0x80 = B&W 0x81 = Sepia 0x82 = Cyanotype 0x83 = Red 0x84 = Yellow 0x85 = Green 0x86 = Blue-green 0x87 = Blue 0x88 = Purple-blue 0x89 = Red-purple 0xff = n/a
57	ToningSaturation	int8u	

Nikon Picture Control Tags

Tag (hex)	Tag (dec)	IFD	Key
0x0000	0	NikonPc	Exif.NikonPc.Version
0x0004	4	NikonPc	Exif.NikonPc.Name
0x0018	24	NikonPc	Exif.NikonPc.Base
0x0030	48	NikonPc	Exif.NikonPc.Adjust
0x0031	49	NikonPc	Exif.NikonPc.QuickAdjust
0x0032	50	NikonPc	Exif.NikonPc.Sharpness
0x0033	51	NikonPc	Exif.NikonPc.Contrast
0x0034	52	NikonPc	Exif.NikonPc.Brightness
0x0035	53	NikonPc	Exif.NikonPc.Saturation
0x0036	54	NikonPc	Exif.NikonPc.HueAdjustment
0x0037	55	NikonPc	Exif.NikonPc.FilterEffect
0x0038	56	NikonPc	Exif.NikonPc.ToningEffect
0x0039	57	NikonPc	Exif.NikonPc.ToningSaturation

The Exiv2 website is generated by reading the tag definitions in the code-base:

```Bash
$ taglist ALL | grep NikonPc | csv -
[NikonPc.Version]   [0] [0x0000]    [NikonPc]   [Exif.NikonPc.Version]  [Undefined] [Ver
[NikonPc.Name]  [4] [0x0004]    [NikonPc]   [Exif.NikonPc.Name] [Ascii] [Name]
[NikonPc.Base]  [24]    [0x0018]    [NikonPc]   [Exif.NikonPc.Base] [Ascii] [Base]
[NikonPc.Adjust]    [48]    [0x0030]    [NikonPc]   [Exif.NikonPc.Adjust]   [Byte]  [Adj
[NikonPc.QuickAdjust]   [49]    [0x0031]    [NikonPc]   [Exif.NikonPc.QuickAdjust]  [Byt
[NikonPc.Sharpness] [50]    [0x0032]    [NikonPc]   [Exif.NikonPc.Sharpness]    [Byte]
[NikonPc.Contrast]  [51]    [0x0033]    [NikonPc]   [Exif.NikonPc.Contrast] [Byte]  [Cor
[NikonPc.Brightness]    [52]    [0x0034]    [NikonPc]   [Exif.NikonPc.Brightness]   [Byt
[NikonPc.Saturation]    [53]    [0x0035]    [NikonPc]   [Exif.NikonPc.Saturation]   [Byt
[NikonPc.HueAdjustment] [54]    [0x0036]    [NikonPc]   [Exif.NikonPc.HueAdjustment]
[NikonPc.FilterEffect]  [55]    [0x0037]    [NikonPc]   [Exif.NikonPc.FilterEffect] [Byt
[NikonPc.ToningEffect]  [56]    [0x0038]    [NikonPc]   [Exif.NikonPc.ToningEffect] [Byt
[NikonPc.ToningSaturation]  [57]    [0x0039]    [NikonPc]   [Exif.NikonPc.ToningSaturati
$
```

I've decided to call a binary element a Field. So we have a class, and vector of fields for a tag, and a map to hold the definitions:

```C++
class Field
{
public:
    Field
    ( std::string name
    , type_e      type
    , uint16_t    start
    , uint16_t    count
    , endian_e    endian = keImage
    )
    : name_  (name)
    , type_  (type)
    , start_ (start)
    , count_ (count)
    , endian_(endian)
    {};
    virtual ~Field() {}
    std::string name  () { return name_   ; }
    type_e      type  () { return type_   ; }
    uint16_t    start () { return start_  ; }
    uint16_t    count () { return count_  ; }
    endian_e    endian() { return endian_ ; }
private:
    std::string name_   ;
    type_e      type_   ;
    uint16_t    start_  ;
    uint16_t    count_  ;
    endian_e    endian_ ;
};
typedef std::vector<Field>   Fields;
typedef std::map<std::string,Fields>  MakerTags;

// global variable
MakerTags makerTags;
...
void init()
{
    nikonDict [ktGroup ] = "Nikon";
...
    nikonDict [ 0x0023 ] = "PictureControl";
...
    makerTags["Exif.Nikon.PictureControl"].push_back(Field("PcVersion"          ,asciiStr
    makerTags["Exif.Nikon.PictureControl"].push_back(Field("PcName"             ,asciiStr
...
    makerTags["Exif.Nikon.PictureControl"].push_back(Field("PcToningEffect"     ,unsigned
    makerTags["Exif.Nikon.PictureControl"].push_back(Field("PcToningSaturation",unsigned
...
}
```

ReportVisitor::visitTag()

```C++
    virtual void visitTag
    ( Io&            io
    , Image&         image
    , uint64_t       address
    , uint16_t       tag
    , type_e         type
    , uint64_t       count
    , uint64_t       offset
    , DataBuf&       buf
    , const TagDict& tagDict
    ) {
        // format the output
        std::ostringstream    os ; os << offset;
        std::string offsetS = typeSize(type)*count > (image.bigtiff_?8:4) ? os.str() :"'
        std::string    name = tagName(tag,tagDict,28);
        std::string   value = buf.toString(type,count,image.endian_);

        if ( printTag(name) ) {
            out() << indent()
                  << stringFormat("%8u | %#06x %-28s |%10s |%9u |%10s | "
                        ,address,tag,name.c_str(),::typeName(type),count,offsetS.c_str()
                  << chop(value,40)
                  << std::endl
            ;
            if ( makerTags.find(name) != makerTags.end() ) {
                for (Field field : makerTags[name] ) {
                    std::string n      = join(groupName(tagDict),field.name(),28);
                    endian_e    endian = field.endian() == keImage ? image.endian() : fi
                    out() << indent()
                          << stringFormat("%8u | %#06x %-28s |%10s |%9u |%10s | "
                                        ,offset+field.start(),tag,n.c_str(),typeName(fi
                          << chop(buf.toString(field.type(),field.count(),endian,field.s
                          << std::endl
                    ;
                }
            }
        }
    } // visitTag
```

The code in visitTag() uses DataBuf.toString() to format the data:

```C++
std::string DataBuf::toString(type_e type,uint64_t count,endian_e endian,uint64_t {
{
    std::ostringstream os;
    std::string        sp;
    uint16_t           size = typeSize(type);
    if ( isTypeShort(type) ){
        for ( uint64_t k = 0 ; k < count ; k++ ) {
            os << sp << ::getShort(*this,offset+k*size,endian);
            sp = " ";
        }
    } else if ( isTypeLong(type) ){
        for ( uint64_t k = 0 ; k < count ; k++ ) {
            os << sp << ::getLong(*this,offset+k*size,endian);
            sp = " ";
        }
    } else if ( isTypeRational(type) ){
        for ( uint64_t k = 0 ; k < count ; k++ ) {
            uint32_t a = ::getLong(*this,offset+k*size+0,endian);
            uint32_t b = ::getLong(*this,offset+k*size+4,endian);
            os << sp << a << "/" << b;
            sp = " ";
        }
    } else if ( isType8Byte(type) ) {
        for ( uint64_t k = 0 ; k < count ; k++ ) {
            os << sp << ::getLong8(*this,offset+k*size,endian);
            sp = " ";
        }
    } else if ( type == kttUByte ) {
        for ( size_t k = 0 ; k < count ; k++ )
            os << stringFormat("%s%d",k?" ":"",pData_[offset+k]);
    } else if ( type == kttAscii ) {
        bool bNoNull = true ;
        for ( size_t k = 0 ; bNoNull && k < count ; k++ )
            bNoNull = pData_[offset+k];
        if ( bNoNull )
            os << binaryToString(offset, (size_t)count);
        else
            os << (char*) pData_+offset ;
    } else {
        os << sp << binaryToString(offset, (size_t)count);
    }

    return os.str();
} // DataBuf::toString
```

Here's the beautiful result on ~/Stonehenge.jpg

```Bash
...book/build $ ./tvisitor -pR ~/Stonehenge.jpg | grep -e PictureControl -e Pc
 286 | 0x0023 Exif.Nikon.PictureControl    | UNDEFINED |       58 |       837 | 0100STAN
 837 | 0x0023 Exif.Nikon.PcVersion         |     ASCII |        4 |           | 0100
 841 | 0x0023 Exif.Nikon.PcName            |     ASCII |       20 |           | STANDARD
 861 | 0x0023 Exif.Nikon.PcBase            |     ASCII |       20 |           | STANDARD
 885 | 0x0023 Exif.Nikon.PcAdjust          |      BYTE |        1 |           | 0
 886 | 0x0023 Exif.Nikon.PcQuickAdjust     |      BYTE |        1 |           | 255
 887 | 0x0023 Exif.Nikon.PcSharpness       |      BYTE |        1 |           | 0
 888 | 0x0023 Exif.Nikon.PcContrast        |      BYTE |        1 |           | 0
 889 | 0x0023 Exif.Nikon.PcBrightness      |      BYTE |        1 |           | 128
 890 | 0x0023 Exif.Nikon.PcSaturation      |      BYTE |        1 |           | 0
 891 | 0x0023 Exif.Nikon.PcHueAdjustment   |      BYTE |        1 |           | 128
 892 | 0x0023 Exif.Nikon.PcFilterEffect    |      BYTE |        1 |           | 255
 893 | 0x0023 Exif.Nikon.PcFilterEffect    |      BYTE |        1 |           | 255
 894 | 0x0023 Exif.Nikon.PcToningSaturat.. |      BYTE |        1 |           | 255
...book/build $
```

Could this be even better? Of course. As always reader, I leave you to send me a patch which will:

1. Test that we only decode bytes read from image.
2. Build and run this on a BigEndian machine (PPC, Sparc)
3. You're welcome to suggest other magic!

TOC

3.6 JpegImage::accept()

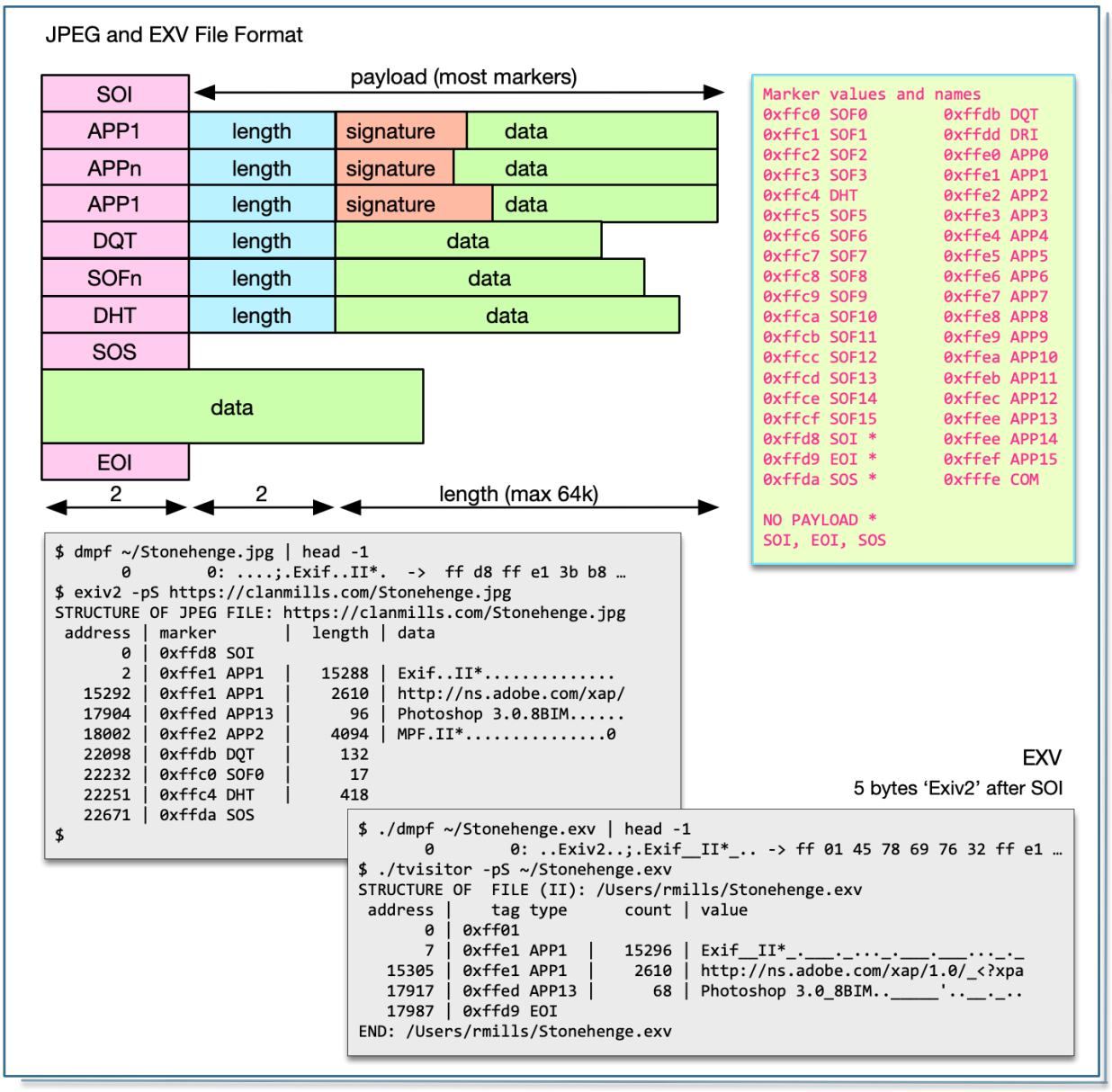
```C++
void JpegImage::accept(Visitor& visitor)
{
    // Ensure that this is the correct image type
    if (!valid()) {
        std::ostringstream os ; os << "expected " << format_ ;
        Error(kerInvalidFileFormat,io().path(),os.str());
    }
    IoSave save(io(),0);
    visitor.visitBegin((*this)); // tell the visitor

    enum                            // kes = Exif State
    { kesNone = 0                   // not reading exif
    , kesAdobe                      // in a chain of APP1/Exif__ segments
    , kesAgfa                       // in AGFA segments of 65535
    }          exifState = kesNone ;
    DataBuf    exif                 ; // buffer to suck up exif data
    uint64_t   nExif     = 0        ; // Count the segments in Exif
    uint64_t   aExif     = 0        ; // Remember address of block0

    DataBuf    XMP                  ; // buffer to suck up XMP
    bool       bExtXMP   = false    ;

    // Step along linked list of segments
    bool     done = false;
    while ( !done ) {
        // step to next marker
        int  marker = advanceToMarker();
        if ( marker < 0 ) {
            Error(kerInvalidFileFormat,io().path());
        }

        size_t      address       = io_.tell()-2;
        DataBuf     buf(48);

        // Read size and signature
        uint64_t    bufRead       = io_.read(buf);
        uint16_t    length        = bHasLength_[marker] ? getShort(buf,0,keBig):0;
        bool        bAppn         = marker >= app0_ && marker <= (app0_ | 0x0F);
        bool        bHasSignature = marker == com_ || bAppn ;
        std::string signature     = bHasSignature ? buf.binaryToString(2, buf.size_ - 2)

        bool        bExif         = bAppn && signature.size() > 6 && signature.find("Exi
        exifState                 = bExif       ? kesAdobe
                                  : (exifState == kesAdobe && length == 65535) ? kesAgfa
                                  : kesNone ;

        if ( exifState ) { // suck up the Exif data
            size_t chop = bExif ? 6 : 0 ;
            exif.read(io_,(address+2)+2+chop,length-2-chop); // read into memory
            if ( !nExif ++ ) aExif = (address+2)+2+chop ;
            if ( length == 65535 && !bExif ) exifState = kesAgfa;
        }

        // deal with deferred Exif metadata
```

```
        if ( !exif.empty() && !exifState )
        {
            IoSave save(io_,aExif);
            Io     file(io_,aExif,exif.size_); // stream on the file
            Io     memory(exif);               // stream on memory buffer
            visitor.visitExif(nExif == 1 ? file :memory ); // tell the visitor
            exif.empty(true)  ; // empty the exif buffer
            nExif     = 0     ; // reset the block counter
        }
        // deal with deferred XMP
        if ( !XMP.empty() && !bAppn ) {
            visitor.visitXMP(XMP); // tell the visitor
            bExtXMP = false ;
            XMP.empty(true) ; // empty the exif buffer
        }
        visitor.visitSegment(io_,*this,address,marker,length,signature); // tell the vis

        if ( bAppn ) {
           ... code to deal with multiple segments for XMP and ICC ...
        }

        // Jump past the segment
        io_.seek(address+2+length); // address is previous marker
        done = marker == eoi_ || marker == sos_ || io().eof();
    } // while !done

    visitor.visitEnd((*this)); // tell the visitor
}  // JpegImage::visitTiff
```

This function is not as simple as TiffImage::accept(). It navigates the chain of segments and calls the visitor appropriately. The function is complicated to deal with Extended JPEG. There are two schemes for dealing with Exif metadata that span more than a single segment.

For the benefit of clarity, I haven't shown the code here which handles Extended XMP. In Exiv2, there is also code to handle ICC profiles which can also span multiple segments.

The way in which extended JPEG is managed is quite simple. A DataBuf is used and as more data is discovered we read from the image source into the DataBuf. After reading consecutive blocks onto memory, we tell the visitor and clear the buffer.


JPEG and EXV File Format
payload (most markers)
SOI
APP1
APPn
APP1
DQT
SOFn
DHT
SOS
EOI
length
signature
data
2
2
length (max 64k)
Marker values and names
0xffc0 SOF0
0xffc1 SOF1
0xffc2 SOF2
0xffc3 SOF3
0xffc4 DHT
0xffc5 SOF5
0xffc6 SOF6
0xffc7 SOF7
0xffc8 SOF8
0xffc9 SOF9
0xffca SOF10
0xffcb SOF11
0xffcc SOF12
0xffcd SOF13
0xffce SOF14
0xffcf SOF15
0xffd8 SOI *
0xffd9 EOI *
0xffda SOS *
0xffdb DQT
0xffdd DRI
0xffe0 APP0
0xffe1 APP1
0xffe2 APP2
0xffe3 APP3
0xffe4 APP4
0xffe5 APP5
0xffe6 APP6
0xffe7 APP7
0xffe8 APP8
0xffe9 APP9
0xffea APP10
0xffeb APP11
0xffec APP12
0xffee APP13
0xffee APP14
0xffef APP15
0xfffe COM
NO PAYLOAD *
SOI, EOI, SOS
$ dmpf ~/Stonehenge.jpg | head -1
0 0: ....;.Exif..II*. -> ff d8 ff e1 3b b8 …
$ exiv2 -pS https://clanmills.com/Stonehenge.jpg
STRUCTURE OF JPEG FILE: https://clanmills.com/Stonehenge.jpg
address | marker | length | data
0 | 0xffd8 SOI
2 | 0xffe1 APP1 | 15288 | Exif..II*.....
15292 | 0xffe1 APP1 | 2610 | http://ns.adobe.com/xap/
17904 | 0xffed APP13 | 96 | Photoshop 3.0.8BIM.....
18002 | 0xffe2 APP2 | 4094 | MPF.II*.....0
22098 | 0xffdb DQT | 132
22232 | 0xffc0 SOF0 | 17
22251 | 0xffc4 DHT | 418
22671 | 0xffda SOS
$
EXV
5 bytes 'Exiv2' after SOI
$ ./dmpf ~/Stonehenge.exv | head -1
0 0: ..Exiv2..;.Exif__II*_.. -> ff 01 45 78 69 76 32 ff e1 …
$ ./tvisitor -pS ~/Stonehenge.exv
STRUCTURE OF FILE (II): /Users/rmills/Stonehenge.exv
address | tag type count | value
0 | 0xff01
7 | 0xffe1 APP1 | 15296 | Exif__II*_.___._...._.___.___..._._
15305 | 0xffe1 APP1 | 2610 | http://ns.adobe.com/xap/1.0/_<?xpa
17917 | 0xffed APP13 | 68 | Photoshop 3.0_8BIM.._____'..__._..
17987 | 0xffd9 EOI
END: /Users/rmills/Stonehenge.exv

TOC

4 Lens Recognition

Lens Recognition is a difficult problem. The lens isn't stored in the metadata. Different manufacturers use different ways to deal with the lens and it's very common that a number such as "368" is used to represent several lenses. Then we have to examine other metadata to make a guess about which lens is being used. Lens recognition has been a time sink on the engineering resources of Team Exiv2. So, I introduced the ~/.exiv2 "Configuration File" in 0.26 to save lots of work and give users an instant way to recognise their lens. You don't need to wait on the release cycles of exiv2 and your distribution. You get it fixed instantly.

In the introduction to this book, I have discussed my proposal for ***M2Lscript*** (pronounce MillsScript). This is my proposal to solve the lens problem. Future Exiv2 Projects

The Configuration File

The configuration file ~/.exiv2 (or %USERPROFILE%\exiv2.ini for Visual Studio Users) may be used to define a lens. For example:

```ini
[nikon]
146=Robin's Sigma Lens
```

If uncertain, exiv2 can display the path:

```bash
696 rmills@rmillsmbp:~/gnu/exiv2/team/book $ exiv2 -vVg config_path  # --verbose -
exiv2 0.27.3
config_path=/Users/rmills/.exiv2
697 rmills@rmillsmbp:~/gnu/exiv2/team/book $
```

Most manufacturers store the LensID (an integer) in their maker notes:

```Bash
$ taglist ALL | grep Lens | grep -ie number -ie id -ie type | csv -
[Photo.LensSpecification]   [42034] [0xa432]    [Photo] [Exif.Photo.LensSpecification]
[Photo.LensModel]   [42036] [0xa434]    [Photo] [Exif.Photo.LensModel]  [Ascii] [This to
[Photo.LensSerialNumber]    [42037] [0xa435]    [Photo] [Exif.Photo.LensSerialNumber]
[CanonCs.LensType]  [22]    [0x0016]    [CanonCs]   [Exif.CanonCs.LensType] [SShort]
[Minolta.LensID]    [268]   [0x010c]    [Minolta]   [Exif.Minolta.LensID]   [Long]  [Ler
[Nikon3.LensType]   [131]   [0x0083]    [Nikon3]    [Exif.Nikon3.LensType]  [Byte]  [Ler
[NikonLd1.LensIDNumber] [6] [0x0006]    [NikonLd1]  [Exif.NikonLd1.LensIDNumber]    [Byt
[NikonLd2.LensIDNumber] [11]    [0x000b]    [NikonLd2]  [Exif.NikonLd2.LensIDNumber]
[NikonLd3.LensIDNumber] [12]    [0x000c]    [NikonLd3]  [Exif.NikonLd3.LensIDNumber]
[OlympusEq.LensType]    [513]   [0x0201]    [OlympusEq] [Exif.OlympusEq.LensType]   [Byt
[OlympusEq.LensSerialNumber]    [514]   [0x0202]    [OlympusEq] [Exif.OlympusEq.LensSeri
[Panasonic.LensType]    [81]    [0x0051]    [Panasonic] [Exif.Panasonic.LensType]   [Asc
[Panasonic.LensSerialNumber]    [82]    [0x0052]    [Panasonic] [Exif.Panasonic.LensSeri
[PentaxDng.LensType]    [63]    [0x003f]    [Pentax]    [Exif.Pentax.LensType]  [Byte]
[Pentax.LensType]   [63]    [0x003f]    [Pentax]    [Exif.Pentax.LensType]  [Byte]  [Ler
[Samsung2.LensType] [40963] [0xa003]    [Samsung2]  [Exif.Samsung2.LensType]    [Short]
[Sony1.LensID]  [45095] [0xb027]    [Sony1] [Exif.Sony1.LensID] [Long]  [Lens identifier
[Sony2.LensID]  [45095] [0xb027]    [Sony1] [Exif.Sony1.LensID] [Long]  [Lens identifier
[SonyMinolta.LensID]    [268]   [0x010c]    [Minolta]   [Exif.Minolta.LensID]   [Long]
[Sony2010e.LensType2]   [6291]  [0x1893]    [Sony2010e] [Exif.Sony2010e.LensType2]  [Sho
[Sony2010e.LensType]    [6294]  [0x1896]    [Sony2010e] [Exif.Sony2010e.LensType]   [Sho
$
```

Manufacturer	Config Section	Metadata
Canon	[canon]	Exif.CanonCs.LensType
Minolta	[minolta]	Exif.Minolta.LensID
Nikon	[nikon]	Exif.NikonLd{1|2|3}.LensIDNumber
Olympus	[olympus]	OlympusEq.LensType
Panasonic	[panasonic]	Exif.Panasonic.LensType
Pentax	[canon]	Exif.Pentax.LensType
Sony	[sony]	Exif.Sony2010e.LensType Exif.Sony2010e.LensType2

Lens in Exif

There are a couple of Exif tags defined in Exif 2.2:

Tag	Type	Description
Exif.Photo.LensSpecification	Rational	Focal length min, max
Exif.Photo.LensModel	Ascii	
Exif.Photo.LensSerialNumber	Ascii	

C++ Lens Recognition

For a discussion about Nikon see: https://github.com/Exiv2/exiv2/issues/743#issuecomment-473409909

TOC

5 I/O in Exiv2

I/O in Exiv2 is achieved using the class BasicIo and derived classes which are:

Name	*Purpose*	*Description*
BasicIo	Abstract	Defines methods such as open(), read(), seek() and others
FileIo	FILE*	Operates on a FILE or memory-mapped file
MemIo	DataBuf_t	Operates on a memory buffer
RemoteIo	Abstract	provides support for url parsing
HttpIo	http:	Simple http 1.1 non-chunked support
FtpIo	ftp:,ftps:	Requires CurlIo
CurlIo	http:,https:	Comprehensive remote I/O support
SshIo	server:path	Requires libssh
StdinIo	-	Read from std-in
Base64Io	data:…..	Decodes ascii encoded binary

You will find a simplified version of BasicIo in tvisitor.cpp in the code that accompanies this book. Io has several constructors. The obvious one is **Io(std::string)** which calls **fopen()**. More subtle is **Io(io,from,size)** which creates a sub-file on an existing stream. This design deals with embedded files. Most metadata is written in a format designated by the standards body and embedded in the file. For example, Exif metadata data is written in Tiff Format and embedded in the file.

The constructor **Io(DataBuf&)** is used to create an in-memory I/O stream. ***DataBuf*** has a **read()** method to binary copy from a stream into memory. As we will see, some subfiles are not contiguous in the image and "chunked" by the image format. For example, JPEG is always chunked into segments of 64k or less. When a subfile has been chunked it is convenient to copy bytes into a buffer from which we can create an Io source.

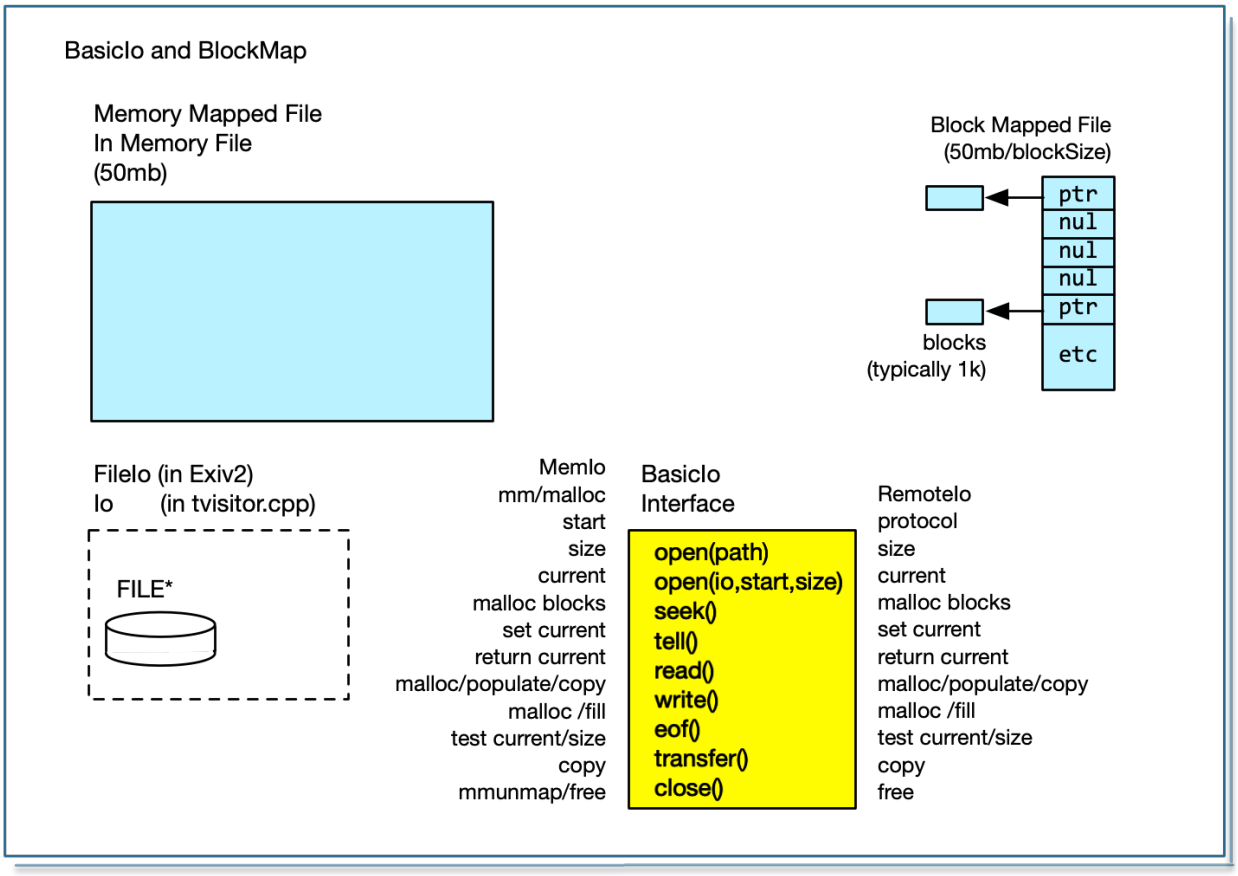
Other metadata standards use a similar design. XMP is embedded XML, an Icc Profile is a major block of technology. Exiv2 knows how to extract, insert, delete and replace an Icc Profile. It knows nothing about the contents of the Icc Profile. With Xmp, Exiv2 uses Adobe's XMPsdk to enable the Xmp data to be modified.

Exiv2 has an abstract RemoteIo object which can read/write on the internet. For http, there is a basic implementation of the http protocol in src/http.cpp. For production use, Exiv2 should be linked with libcurl. The reason for providing a "no thrills" implementation of http was two fold. Firstly, it enabled the project to proceed rapidly without learning the curl API. Secondly, I wanted all versions of the exiv2 command-line to have http support as I thought it would be useful for testing as we could store video and other large files remotely.

The MemIo class enables memory to be used as a stream. This is fast and convenient for small temporary files. When memory mapped files are available, FileIo uses that in preference to FILE*. When the project started in

2004, memory-mapped files were not provided on some legacy platforms such as DOS. Today, all operating systems provide memory mapped files. I've never heard of Exiv2 being used in an embedded controller, however I'm confident that this is feasible. I've worked on embedded controllers with no operating system and only a standard "C" io library. Exiv2 can be built for such a device.

Most camera manufacturers are large corporations. I'm sure they have their own firmware to handle Exif metadata. However, the world of photography has an ever growing band of start-ups making amazing devices such as Go-Pro. One day I'll hear that somebody is cycling around on top of Mt Everest with Exiv2 running on top of their head! One of our users is an astronomer at NASA. I've never heard that Exiv2 has flown in space, however one day it might. I will say with pride that Exiv2 is out of this world!



Using memory mapped files

When available, Exiv2 uses memory mapped files. This is not a good idea for several reasons. Firstly, image editing applications can sit for days with a file open. For example, a GIMP user may open a file on Monday and it may be still be open several days later. In the meanwhile things have changed on the network. Secondly, memory mapped files on Windows are locked by the operating system. This causes problems with the virus checker. Thirdly, it's possible for another application to modify a file which is memory mapped. Exiv2 has copied the metadata into memory and can have stale/obsolete data.

The reason for using memory mapped files was for the convenience of converting offsets into memory addresses. Imperial College have 90GByte Tiffs from medical imaging products. We have to map 90GBytes. And it gets worse, some file handlers allocate and copy the file before processing. As we can see in tvisitor.cpp,

it's possible to navigate the metadata in huge files with very little I/O. Memory Mapped files for metadata processing have turned out to have sad consequences.

Writing Files

Exiv2 is very reliable at writing files which conform to standards. The way in which this is achieved is to by calling image->writeMetata() which delegates to the handlers writeMetata().

Because the handler understands the structure of the image, he writes a temporary in memory copy of the image. It proceeds to parse the image and copy the data to the temporary file. When it arrives at each of the four metadata blocks (Exif, ICC, IPTC and XMP) it calls the serializer to create a buffer of data which is injected into the temporary image. When it arrives the EOF on the original file, if no error has been detected it calls io->transfer() on the temporary image. The operation transfer() copies the bytes from the temporary stream to the permanent file.

This method is very robust and reliable. For very very files (for example, 100GB medical imaging file), this places huge demands on memory. For remote file, it requires every byte from the remote location to be copied to the temporary file and subsequently transferred back to the remote location. One day a project will be undertaken to stress test remote IO on HUGE files and more will be understood about the performance and optimisation that can be undertaken.

Intrusive and NonIntrusive Write Mode

When Exiv2 rewrites an image, it determines the writeMode to determines the writeMode which are:

1. Non-intrusive The metadata is updated in-place. For performance reasons, this the default as it means that metadata can be updated by modifying a bytes in the original file. For example, a common metadata edit is to change the date in Exif.Image.DateTime. Non-intrusive write mode is designed to ensure this is performed very quickly.

2. Intrusive The metadata is totally re-written in memory. This always occurs if there are any changes in the makernote. It will always occur if any tag edited tag requires more storage than in the original file.

Write Mode is really clever, however it's scope is limited to writing Tiff images (and therefore similar Raw formats such as DNG, CR2 and NEF), only a small part of the file is written as a Tiff (the Exif metadata) and the image handler must use the io()->transfer() mechanism discussed above.

Using a Block Map to track changes to the file.

In Chapter 5, I discuss the use of a block map to track small areas of the file which are in use. I'm confident that architecture could be developed to vastly reduce the I/O involved in updating the metadata in a file. 5. I/O in Exiv2

TOC

6 Image Previews

I don't know much about the image previews. Previews are usually JPEG encoded and have no metadata. Exiv2 has no code to edit previews in images. About all that I know about previews is that the library finds them and creates a vector of thumbnails. Like most of Andreas' code, the Preview code works well and has seldom required attention.

There are significant challenges in finding the previews as manufacturers use a variety of techniques. In particular, they often store an offset to a preview in a makernote, or some other devious location. In consequence, it's almost impossible to re-write the file without the risk of losing the preview. This problem is compounded by the JPEG 64k limit in a single segment. Digital Cameras and Smart Phones are now a huge business and JPEG is the most popular image format. Regrettably, JPEG is a 30 year old standard which was conceived when dinosaurs roamed the earth. A global agreement to support Adobe's ***ad-hoc*** JPEG extension could easily address this issue. The inertia of the industry is colossal.

I find it incredible, yet unsurprising, that in an industry which talks about inovation and development can be so resistant to change. Small changes that would serve their industry. A friend of mine in Silicon Valley sat on a standards committee for video encoding. He told me about a meeting in which something absurd was proposed and he decided, although he seldom spoke, to speak against it. He couldn't let the issue pass. He said to me with a war-comic German accent. *"So, I took zee lugar and I aimed carefully and I squeezed zee trigger. The bullet flew fast and straight and hit me between the eyes."*. The opposition he encountered was breathtaking. Craziness ends up in standards because of the politics of the Standards Committee. So that's how we end up with several different designs to enable Exif, ICC and XMP data to be chunked in a JPEG. And the mess with Lens Recognition. And the mess with hundreds of similar yet different image formats.

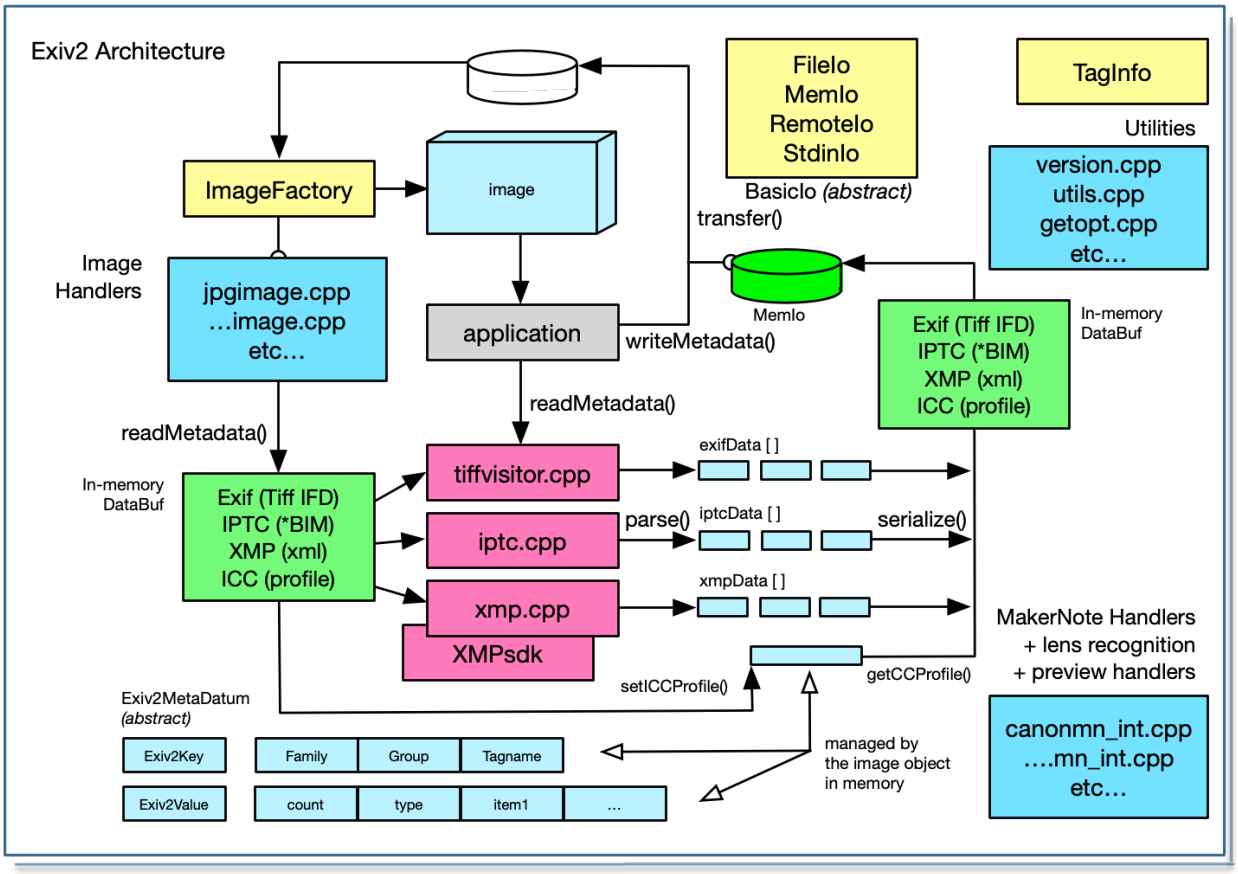
```Bash
786 rmills@rmillsmm-local:~/temp/foo $ cp ~/Stonehenge.jpg .
787 rmills@rmillsmm-local:~/temp/foo $ exiv2 -ep --verbose Stonehenge.jpg
File 1/1: Stonehenge.jpg
Writing preview 1 (image/jpeg, 160x120 pixels, 10837 bytes) to file ./Stonehenge-preview
788 rmills@rmillsmm-local:~/temp/foo $ exiv2 -pS ./Stonehenge-preview1.jpg
STRUCTURE OF JPEG FILE: ./Stonehenge-preview1.jpg
 address | marker       |  length | data
       0 | 0xffd8 SOI
       2 | 0xffdb DQT   |     132
     136 | 0xffc0 SOF0  |      17
     155 | 0xffc4 DHT   |     418
     575 | 0xffda SOS
789 rmills@rmillsmm-local:~/temp/foo $
```

dmpf.cpp finds it. So, we know it is 4448 bytes into the file and the Exif Tiff is 15288 bytes and begins at 12. So it's in there, but where? I don't know. More research needed.

```Bash
.../book/build $ ./dmpf ~/temp/foo/Stonehenge-preview1.jpg | head -1
       0        0: ...._._.........................  ->  ff d8 ff db 00 84 00 ...
.../book/build $ ./dmpf ~/temp/foo/Stonehenge.jpg | grep 'ff d8 ff db 00 84'
  0x1160     4448: __.___....._._...................  ->  00 00 01 00 00 00 ff d8 ff db 0
.../book/build $
```

TOC

7 Exiv2 Architecture



The Exiv2 API is documented here: https://exiv2.org/doc/ The API is in the Namespace Exiv2. The Namespace Exiv2::Internal should never be used by application programs and is not revealed to via <exiv2/exiv2.hpp>. As there are around 300 classes and 3000+ entry points, it's not possible to discuss the API in detail here. Instead I will discuss a typical short application: samples/exifprint.cpp. My aim here is to explain how to use the exiv2 library and provide a high-level overview of how the library operates. The doxygen generated API Documentation is very good and the code is well laid out and documented.

TOC

7.1 API Overview

1) The Image Handlers

This code understands the structure of image files. The structure is explained in Chapter 1 of this book.

2) The MetaData parsers

This code understands the structure of the different metadata standards. The structure is explained in Chapter 2 of this book.

3) Manufacturer's MakerNote handlers

All the manufacturers use variations of the TIFF/IFD format in their makernote. The maker note is parsed by the TiffParser. The presentation and interpretation of the makenote is handled here. In particular the lens recognition and preview image handling is dealt with in this code. 4. Lens Recognition. 6. Image Previews

4) TagInfo

This code has definitions for thousands of Exif tags and about 50 IPTC Tags. Xmp metadata is handled by the XMPsdk. As XMP is Extensible, it doesn't have a database of known tags. Tags are discussed in detail in Chapter 6 of this book.

5) BasicIo

This code is responsible for all I/O and is explained in Chapter 5 of this book.

6) Utility and Platform Code

There are utility functions such as ascii 64 encode/decode. There are platform specific functions which manage interaction with the platform operating system.

7) The Image Object and Image Factory

Applications obtain access to an image object via the Image Factory. The application is expected to call readMetadata() which causes the image handler to locate metadata and passes it to the metadata handlers for conversion to a metadatum vector. Metadatum elements are key/value pairs. Metadatum can be manipulated in memory or presented to the user. If the metadata has been modified, the application should call writeMetadata() which will cause the reading process to be reversed. The metadatum vector is serialized and the file is rewritten by the Image Handler.

8) Sample Code and Test Harness

This is discussed here: 8. Test Suite

TOC

7.2 Typical Sample Application

```C++
// ********************************************************************* -*- C++ -*-
// exifprint.cpp
// Sample program to print the Exif metadata of an image
// g++ -std=c++98 exifprint.cpp -I/usr/local/include -L/usr/local/lib -lexiv2 -o exifpr

#include <exiv2/exiv2.hpp>

#include <iostream>
#include <iomanip>
#include <cassert>

int main(int argc, const char* argv[])
{
    try {
        Exiv2::XmpParser::initialize();
        ::atexit(Exiv2::XmpParser::terminate);

        const char* prog = argv[0];
        const char* path = argv[1];

        if (argc != 2) {
            std::cout << "Usage: " << prog << " [ path | --version ]" << std::endl;
            return 1;
        }

        if ( strcmp(path,"--version") == 0 ) {
            exv_grep_keys_t keys;
            Exiv2::dumpLibraryInfo(std::cout,keys);
            return 0;
        }

        Exiv2::Image::AutoPtr image = Exiv2::ImageFactory::open(path);
        assert(image.get() != 0);
        image->readMetadata();

        Exiv2::ExifData &exifData = image->exifData();
        if (exifData.empty()) {
            std::string error("No Exif data found in file");
            throw Exiv2::Error(Exiv2::kerErrorMessage, error);
        }

        for (Exiv2::ExifData::const_iterator i = exifData.begin(); i != exifData.end();
            std::cout << i->key() << " -> " << i->toString() << std::endl;
        }

        return 0;
    } catch (Exiv2::Error& e) {
        std::cout << "Caught Exiv2 exception '" << e.what() << "'\n";
        return -1;
    }
}
```

Include file

Only include the file <exiv2/exiv2.hpp>. Do not include individual exiv2 include files because Team Exiv2 may remove or add include files. By only including <exiv2/exiv2.hpp>, you are insulated from changes to the dependency and existence of individual include files.

```C++
#include <exiv2/exiv2.hpp>

#include <iostream>
...
```

Initializing the library

You do not need to initialize the exiv2 library. However you have to initialize XMPsdk.

```C++
        Exiv2::XmpParser::initialize();
        ::atexit(Exiv2::XmpParser::terminate);
```

Opening an image to read metadata

Use the ImageFactory to open the image on a path. Verify that the image is good, then call readMetadata()

```C++
        Exiv2::Image::AutoPtr image = Exiv2::ImageFactory::open(path);
        assert(image.get() != 0);
        image->readMetadata();

        Exiv2::ExifData &exifData = image->exifData();
        if (exifData.empty()) {
            std::string error("No Exif data found in file");
            throw Exiv2::Error(Exiv2::kerErrorMessage, error);
        }
```

Stepping through the metadata

The metadata is stored in an STL vector which you can step in the conventional way:

```C++
        for (Exiv2::ExifData::const_iterator i = exifData.begin(); i != exifData.end(
            std::cout << i->tag() << " -> " << i->toString() << std::endl;
        }
```

Each enumerated item is of the Exiv2::Exifdatum: https://exiv2.org/doc/classExiv211Exifdatum.html for which there are many getter functions such as key(), familyName(), count() and toString().

Making Changes to the Metadata

The application samples/addmodel.cpp *(add modify delete)* illustrates how to manipulate metadata: https://exiv2.org/doc/addmoddel_8cpp-example.html. Frequently, you can add/modify metadata directly with code such as:

```C++
        exifData["Exif.Image.Model"] = "Test 1"
```

To delete metadata, you have to locate the key in the ExifData vector and erase it from the vector.

```C++
Exiv2::ExifKey            key("Exif.Photo.DateTimeOriginal");
Exiv2::ExifData::iterator pos = exifData.findKey(key);
if (pos == exifData.end()) {
    throw Exiv2::Error(Exiv2::kerErrorMessage, "Key not found");
}
exifData.erase(pos);
```

Writing modified metada to storage

When you modify metadata using the variable *image*, you are only changing it in memory. You commit the changes to storage when you call image->writeMetadata().

```C++
image->writeMetadata()
```

The image will be automatically closed when image goes out of scope.

TOC

7.3 The EasyAccess API

Exiv2 provides a collection of functions to simplify searching for Exif metadata. This is described in detail here: https://github.com/Exiv2/exiv2/wiki/EasyAccess-API

A typical use case is:

```C++
Exiv2::ExifData::const_iterator metadata = Exiv2::whiteBalance(exifData);
if ( metadata != exifData.end() ) {
    metadata->write(std::cout, &exifData);
}
```

The following EasyAccess Selector Functions are provided:

a-e	e-f	i-m	m-s	s-w
afPoint	exposureMode	imageQuality	meteringMode	serialNumber
apertureValue	exposureTime	isoSpeed	model	sharpness
brightnessValue	flash	lensName	orientation	shutterSpeedValue
contrast	flashBias	lightSource	saturation	subjectArea
dateTimeOriginal	flashEnergy	macroMode	sceneCaptureType	subjectDistance
exposureBiasValue	fNumber	make	sceneMode	whiteBalance
exposureIndex	focalLength	maxApertureValue	sensingMethod	

TOC

7.4 Listing the API

You can get a list of the API with a command such as:

```Bash
$ nm -g --demangle build/lib/libexiv2.dylib | grep ' T ' | grep Exiv2
```

You can refine that further to discover all the "free" functions of the library which are:

```Bash
$ nm -g --demangle build/lib/libexiv2.dylib | grep ' T ' | grep Exiv2 | grep -v tr
0000000000073050 T enforce(bool, Exiv2::ErrorCode)
0000000000107750 T Exiv2::exvGettext(char const*)
000000000012c360 T Exiv2::testVersion(int, int, int)
0000000000072190 T Exiv2::base64decode(char const*, char*, unsigned long)
0000000000071dc0 T Exiv2::base64encode(void const*, unsigned long, char*, unsigned long)
000000000012bd00 T Exiv2::versionNumber()
000000000012bd10 T Exiv2::versionString()
0000000000073fc0 T Exiv2::getProcessPath()
00000000001057f0 T Exiv2::floatToRationalCast(float)
000000000012bfd0 T Exiv2::versionNumberHexString()
0000000000108460 T int Exiv2::gcd<int>(int, int)
0000000000052d80 T Exiv2::errMsg(int)
0000000000071750 T Exiv2::getEnv(int)
00000000000718b0 T Exiv2::to_hex(char)
000000000012c340 T Exiv2::version()
00000000001075f0 T Exiv2::exifTime(char const*, tm*)
00000000000718e0 T Exiv2::from_hex(char)
0000000000072d10 T Exiv2::strError()
0000000000071b80 T Exiv2::urldecode(char const*)
0000000000071980 T Exiv2::urlencode(char const*)
```

You could refine that to reveal the members of class Exiv2::ExifKey:

```Bash
$ nm -g --demangle build/lib/libexiv2.dylib | grep ' T ' | grep ' Exiv2::ExifKey'
00000000000f9f30 T Exiv2::ExifKey::setIdx(int)
00000000000f9a70 T Exiv2::ExifKey::ExifKey(Exiv2::TagInfo const&)
00000000000f9ba0 T Exiv2::ExifKey::ExifKey(std::__1::basic_string...
00000000000f9d00 T Exiv2::ExifKey::ExifKey(Exiv2::ExifKey const&)
00000000000f97f0 T Exiv2::ExifKey::ExifKey(unsigned short, std::__1::basic_string...
00000000000f9830 T Exiv2::ExifKey::ExifKey(Exiv2::TagInfo const&)
00000000000f9aa0 T Exiv2::ExifKey::ExifKey(std::__1::basic_string<char...
00000000000f9bd0 T Exiv2::ExifKey::ExifKey(Exiv2::ExifKey const&)
00000000000f94b0 T Exiv2::ExifKey::ExifKey(unsigned short, std::__1::basic_string...
00000000000f9da0 T Exiv2::ExifKey::~ExifKey()
00000000000f9d80 T Exiv2::ExifKey::~ExifKey()
00000000000f9d30 T Exiv2::ExifKey::~ExifKey()
00000000000f9e00 T Exiv2::ExifKey::operator=(Exiv2::ExifKey const&)
00000000000f9fd0 T Exiv2::ExifKey::familyName() const
00000000000fa240 T Exiv2::ExifKey::defaultTypeId() const
00000000000fa4a0 T Exiv2::ExifKey::idx() const
00000000000f9f70 T Exiv2::ExifKey::key() const
00000000000fa2b0 T Exiv2::ExifKey::tag() const
00000000000fa2e0 T Exiv2::ExifKey::clone() const
00000000000f7880 T Exiv2::ExifKey::ifdId() const
00000000000fa430 T Exiv2::ExifKey::clone_() const
00000000000fa180 T Exiv2::ExifKey::tagDesc() const
00000000000fa070 T Exiv2::ExifKey::tagName() const
00000000000fa0c0 T Exiv2::ExifKey::tagLabel() const
00000000000fa010 T Exiv2::ExifKey::groupName() const
$
```

TOC

7.5 Function Selectors

A common pattern in the Exiv2 code is the table/function pattern.

Fuction	Purpose
cfgSelFct	determine which cfg + def of a corresponding array-set to use.
ConvertFct	Convert between two keys
CrwEncodeFct CrwDecode	Encoding/Decoding for CRW
CryptFct	Cipher/Decipher Data
EncoderFct DecoderFct	Encoding/Decoding functions for Exif, Iptc and XMP data
EasyAccessFct	See 7.3 The EasyAccess API
InstanceFct	See Other Exiv2 Classes Creates new Image instances
LensIdFct	Convert lens ID to lens name
NewMnFct	Makernote create function for images and groups
NewTiffCompFct	Creates TiffGroupStruct's
PrintFct	Print the "translated" value of data
TagListFct	Get a function to return an array of tags

It's not really clear to me why this is done and it feels like C++ being implemented in C.

TOC

7.6 Tags in Exiv2

The following test program is very useful for understanding tags:

```Bash
$ taglist --help
Usage: taglist [--help]
            [--group name|
             Groups|Exif|Canon|CanonCs|CanonSi|CanonCf|Fujifilm|Minolta|Nikon1|Nikon2|Nik
             Panasonic|Pentax|Sigma|Sony|Iptc|
             dc|xmp|xmpRights|xmpMM|xmpBJ|xmpTPg|xmpDM|pdf|photoshop|crs|tiff|exif|aux|ip
            ]
Print Exif tags, MakerNote tags, or Iptc datasets
```

How Tags are organised:

Element	Definition	Example
Tag	Family.Group.TagName	Exif.Image.Model
Family	Exif or Iptc or Xmp	
Group	There are 106 groups Further discussed below.	Minolta MinoltaCs5D…
TagName	Can be almost anything	TagName is a sub-part of a Group

```Bash
$ taglist MinoltaCsNew | csv -
[ExposureMode]  [1] [0x0001]    [MinoltaCsNew]  [Exif.MinoltaCsNew.ExposureMode]    [Lon
[FlashMode] [2] [0x0002]    [MinoltaCsNew]  [Exif.MinoltaCsNew.FlashMode]   [Long]  [Flo
...
[FlashMetering] [63]    [0x003f]    [MinoltaCsNew]  [Exif.MinoltaCsNew.FlashMetering]
$
```

There isn't a tag Exif.MinoltaCsNew.ISOSpeed. There is a Exif.MinoltaCSNew.ISO

```Bash
$ taglist all | grep ISOSpeed$
Photo.ISOSpeed
PanasonicRaw.ISOSpeed
CanonCs.ISOSpeed
CanonSi.ISOSpeed
Casio2.ISOSpeed
MinoltaCs5D.ISOSpeed
MinoltaCs7D.ISOSpeed
Nikon1.ISOSpeed
Nikon2.ISOSpeed
Nikon3.ISOSpeed
Olympus.ISOSpeed
Olympus2.ISOSpeed
Sony1MltCs7D.ISOSpeed
```

```
$ taglist all | grep \\.ISO$
Casio.ISO
Casio2.ISO
MinoltaCsOld.ISO
MinoltaCsNew.ISO
NikonIi.ISO
NikonSiD300a.ISO
NikonSiD300b.ISO
NikonSi02xx.ISO
NikonSi01xx.ISO
PentaxDng.ISO
Pentax.ISO
Samsung2.ISO
Sony1MltCsOld.ISO
Sony1MltCsNew.ISO
```

You can use the program exifvalue to look for a tag in a file. If the tag doesn't exist in the file, it will report

"value not set":

```
$ exifvalue ~/Stonehenge.jpg Exif.MinoltaCsNew.ISO
Caught Exiv2 exception 'Value not set'
$
```

Bash

If the tag is not known, it will report 'Invalid tag':

```
$ exifvalue ~/Stonehenge.jpg Exif.MinoltaCsNew.ISOSpeed
Caught Exiv2 exception 'Invalid tag name or ifdId `ISOSpeed', ifdId 37'
$
```

Bash

Is there a way to report every tag known to exiv2? Yes. There are 5430 known tags:

```
$ for group in $(taglist Groups); do for tag in $(taglist $group | cut -d, -f 1) ;
Image.ProcessingSoftware
Image.NewSubfileType
Image.SubfileType
Image.ImageWidth
...
$ for group in $(taglist Groups); do for tag in $(taglist $group | cut -d, -f 1) ; do ec
    5430    5430  130555
$
```

Bash

Let's discuss why there are 106 groups. There are about 10 camera manufacturers (Canon, Minolta, Nikon etc) and they use the tag Exif.Photo.MakerNote to store data in a myriad of different (and proprietary standards).

```
$ exifvalue ~/Stonehenge.jpg Exif.Photo.MakerNote
78 105 107 111 110 0 2 ...
```

Bash

Exiv2 has code to read/modify/write makernotes. All achieved by reverse engineering. References on the web site. https://exiv2.org/makernote.html

The MakerNote usually isn't a simple structure. The manufacturer usually has "sub-records" for Camera Settings (Cs), AutoFocus (Af) and so on. Additionally, the format of the sub-records can evolve and change with different models from the manufacturer. For example (as above):

```
$ taglist Groups | grep Minolta
Minolta
MinoltaCs5D
MinoltaCs7D
MinoltaCsOld
MinoltaCsNew
SonyMinolta
$
```

Bash

So, Minolta have 6 "sub-records". Other manufacturers have more. Let's say 10 manufacturers have an average of 10 "sub-records". That's 100 groups.

TOC

TagInfo

Tag definitions are not constant. A tag is simply an uint16_t. The Tiff Standard specifies about 50 tags. Anybody creating an IFD can use the same tag number for different purposes. The Tiff Specification says *"TIFF readers must safely skip over these fields if they do not understand or do not wish to use the information."*.

Exif has to recognise every tag, however it does not need to understand it. In a tiff file, the pixels are located using the tag StripOffsets. We report StripOffsets, however we don't read pixel data. When you execute the exiv2 command-line utility with the argument –unknown, they will be listed. For example:

```Bash
$ exiv2 --unknown -pe ~/Stonehenge.jpg
...
Exif.Nikon3.0x002d                           Short       3  512 0 0
...
$ tvisitor -pUR ~/Stonehenge.jpg
...
          346 | 0x002d Exif.Nikon.0x2d              |     SHORT |         3 |       1499
...
$

Most MakerNotes contain tags which are unknown to Exiv2.


If the user wishes to recover data such as the pixels, it is possible to do this with th

The known tags in Exiv2 are defined in the TagInfo structure.

```cpp
const TagInfo Nikon1MakerNote::tagInfo_[] = {
    TagInfo(0x0001, "Version", N_("Version"),
            N_("Nikon Makernote version"),
               nikon1Id, makerTags, undefined, -1, printValue),
    TagInfo(0x0002, "ISOSpeed", N_("ISO Speed"),
            N_("ISO speed setting"),
            nikon1Id, makerTags, unsignedShort, -1, print0x0002),

const TagInfo CanonMakerNote::tagInfo_[] = {
        TagInfo(0x0000, "0x0000", "0x0000", N_("Unknown"), canonId, makerTags, unsigned
        TagInfo(0x0001, "CameraSettings", N_("Camera Settings"), N_("Various camera set
        TagInfo(0x0002, "FocalLength", N_("Focal Length"), N_("Focal length"), canonId,

const TagInfo gpsTagInfo[] = {
    TagInfo(0x0000, "GPSVersionID", N_("GPS Version ID"),
            N_("Indicates the version of <GPSInfoIFD>. The version is given "
            "as 2.0.0.0. This tag is mandatory when <GPSInfo> tag is "
            "present. (Note: The <GPSVersionID> tag is given in bytes, "
            "unlike the <ExifVersion> tag. When the version is "
            "2.0.0.0, the tag value is 02000000.H)."),
            gpsId, gpsTags, unsignedByte, 4, print0x0000),
    TagInfo(0x0001, "GPSLatitudeRef", N_("GPS Latitude Reference"),
            N_("Indicates whether the latitude is north or south latitude. The "
            "ASCII value 'N' indicates north latitude, and 'S' is south latitude."),
            gpsId, gpsTags, asciiString, 2, EXV_PRINT_TAG(exifGPSLatitudeRef)),
```

As we can see, tag == 1 in the Nikon MakerNotes is Version. In Canon MakerNotes, it is CameraSettings. IN GPSInfo it is GPSLatitudeRef. We need to use the appropriate tag dictionary for the IFD being parsed. The tag
```

0xffff in tagDict in tvisitor.cpp stores the group name of the tags.

TOC

Tag Encryption

Exiv2 does not decrypt or encrypt any data. The sony tag 0x9402 is used to store FocusPosition and the data is ciphered. It's a simple cipher and the code was provided by Phil Harvey. The code is discussed in the detail in the issue referenced in the code below.

The ArrayCfg structure allows any tag to be decrypted by readMetadata() andn encrypted by writeMetadata(). I believe sony tag 0x9402 is the only tag that takes advantage of this feature of the TiffVisitor.

```C++
// https://github.com/Exiv2/exiv2/pull/906#issuecomment-504338797
static DataBuf sonyTagCipher(uint16_t /* tag */, const byte* bytes, uint32_t size, Tiff(
{
    DataBuf b(bytes,size); // copy the data

    // initialize the code table
    byte  code[256];
    for ( uint32_t i = 0 ; i < 249 ; i++ ) {
        if ( bDecipher ) {
            code[(i * i * i) % 249] = i ;
        } else {
            code[i] = (i * i * i) % 249 ;
        }
    }
    for ( uint32_t i = 249 ; i < 256 ; i++ ) {
        code[i] = i;
    }

    // code byte-by-byte
    for ( uint32_t i = 0 ; i < size ; i++ ) {
        b.pData_[i] = code[bytes[i]];
    }

    return b;
}
```

TOC

7.7 Tag Decoder

This is a story in two parts. Firstly, we have to find metadata which is formatted as a Tiff Entry and I call that the ***Metadata Decoder***. Some tags are encoded in binary which must be decoded. I call that the ***Binary Tag Decoder***.

Metadata Decoder

Please read: #988

This PR uses a decoder listed in TiffMappingInfo to decode Exif.Canon.AFInfo. The decoding function

"manufactures" Exif tags such as Exif.Canon.AFNumPoints from the data in Exif.Canon.AFInfo. These tags must never be written to file and are removed from the metadata in exif.cpp/ExifParser::encode().

Three of the tags created (AFPointsInFocus,AFPointsSelected, AFPrimaryPoint) are bitmasks. As the camera can have up to 64 focus points, the tags are a 64 bit mask to say which points are active. The function printBitmask() reports data such as 1,2,3 or (none).

This decoding function decodeCanonAFInfo() added to TiffMappingInfo manufactures the new tags. Normally, tags are processed by the binary tag decoder and that approach was taken in branch fix981_canonAf. However, the binary tag decoder cannot deal with AFInfo because the size of some metadata arrays cannot be determined at compile time.

We should support decoding AFInfo in 0.28, however we should NOT auto-port this PR. We can avoid having to explicitly delete tags from the metadata before writing by adding a "read-only" flag to TagInfo. This would break the Exiv2 v0.27 API and has been avoided. There is an array in decodeCanonAFInfo() which lists the "manufactured" tags such as Exif.Canon.AFNumPoints. In the Exiv2 v0.28 architecture, a way might be designed to generate that data at run-time.

Binary Tag Decoder

Please read: #900

There is a long discussion in #646 about this issue and my investigation into how the makernotes are decoded.

History

The tag for Nikon's AutoFocus data is 0x00b7

Nikon encode their version of tag in the first 4 bytes. There was a 40 byte version of AutoFocus which decodes as Exif.NikonAf2.XXX. This new version (1.01) is 84 bytes in length and decoded as Exif.NikonAf22.XXX.

The two versions (NikonAF2 and NikonAF22) are now encoded as a set with the selector in tiffimage_int.cpp

```C++
    extern const ArraySet nikonAf2Set[] = {
        { nikonAf21Cfg, nikonAf21Def, EXV_COUNTOF(nikonAf21Def) },
        { nikonAf22Cfg, nikonAf22Def, EXV_COUNTOF(nikonAf22Def) },
    };
```

The binary layout of the record is defined in tiff image_int.cpp. For example, AF22 is:

```C++
    extern const ArrayCfg nikonAf22Cfg = {
        nikonAf22Id,      // Group for the elements
        littleEndian,     // Byte order
        ttUndefined,      // Type for array entry
        notEncrypted,     // Not encrypted
        false,            // No size element
        true,             // Write all tags
        true,            // Concatenate gaps
        { 0, ttUnsignedByte,  1 }
    };
    //! Nikon Auto Focus 22 binary array - definition
    extern const ArrayDef nikonAf22Def[] = {
        {  0, ttUndefined,     4 }, // Version
        {  4, ttUnsignedByte,  1 }, // ContrastDetectAF
        {  5, ttUnsignedByte,  1 }, // AFAreaMode
        {  6, ttUnsignedByte,  1 }, // PhaseDetectAF
        {  7, ttUnsignedByte,  1 }, // PrimaryAFPoint
        {  8, ttUnsignedByte,  7 }, // AFPointsUsed
        { 70, ttUnsignedShort, 1 }, // AFImageWidth
        { 72, ttUnsignedShort, 1 }, // AFImageHeight
        { 74, ttUnsignedShort, 1 }, // AFAreaXPosition
        { 76, ttUnsignedShort, 1 }, // AFAreaYPosition
        { 78, ttUnsignedShort, 1 }, // AFAreaWidth
        { 80, ttUnsignedShort, 1 }, // AFAreaHeight
    };
```

The two versions of the data are encoded in tiffimage_int.cpp

```C++
        { Tag::root, nikonAf21Id,      nikon3Id,         0x00b7    },
        { Tag::root, nikonAf22Id,      nikon3Id,         0x00b7    },
```

Binary Selector

The code to determine which version is decoded is in tiffimage_int.cpp

```C++
        {    0x00b7, nikon3Id,         EXV_COMPLEX_BINARY_ARRAY(nikonAf2Set, nikonA
```

When the tiffvisitor encounters 0x00b7, he calls nikonAf2Selector() to return the index of the binary array to be used. By default it returns 0 (the existing `nikonAf21Id`). If the tag length is 84, he returns 1 for `nikonAf21Id`

```C++
    int nikonAf2Selector(uint16_t tag, const byte* /*pData*/, uint32_t size, TiffC
    {
        int result = tag == 0x00b7 ? 0 : -1 ;
        if (result > -1 && size == 84 ) {
            result = 1;
        }
        return result;
    }
```

The decoder

```C++
EXV_CALL_MEMBER_FN(*this, decoderFct)(object);
```

This function understands how to decode byte-by-byte from `const ArrayDef` into the Exiv2 tag/values such as Exif.NikonAF22.AFAreaYPosition which it stores in the ExifData vector.

TOC

7.8 TiffVisitor

Exiv2 has an abstract TiffVisitor class, and the following concrete visitors:

Class	*Derived from*	**Purpose**	**Description**
class TiffReader	TiffVisitor	Reads metadata into memory	image->readMetadata()
class TiffFinder	TiffVisitor	Search an IFD	Finds the "Make" tag 0x010f in IFD0
class TiffDecoder	TiffVisitor	Decodes metadata	To be written
class TiffEncoder	TiffVisitor	Encodes metadata	To be written
class TiffCopier	TiffVisitor	Visits file and copies to a new file	image->writeMetadata()

TiffVisitor is the "beating heart" of Exiv2. It is both ingeneous and very difficult to understand. Although I've worked on the Exiv2 code for more than 12 years, it is only in the process of writing this book that I have come to an (*incomplete)* understanding of its design.

TiffVisitor is actually a state machine with a stack. The code pushes an initial object on the stack and procedes to process the element on top of stack until empty. Some tags, such as a makernote push objects on the stack. Reaching the end of an object, pops the stack. There is a "go" flag to enable the visitor to abort. The TiffReader creates a vector of objects which are post-processed to create the metadata.

The code in tvisitor.cpp uses the run-time stack to maintain state. It simply recursively invokes the code necessary to decode a tag. Much simpler and easier to understand. When debugging, you can examine the stack to understand how/why you arrived at a metadata key/value. TiffVisitor does invoked code recursively and that can be seen in the debugger. However, the branching is achieved using Function Selectors and much more difficult to understand.

IfdId and Group

This is a collection of more than 100 values which are used to track the groups in the MetaData. For example ifdIdNotSet is an initial defined state (with no metadata), ifd0Id represents IFD0, exifId the Exif IFD and so on. There are over one hundred groups (as explained in the man page) to deal with every maker and their binary encoded metadata. The GroupID is a subset of IfdId.

Tag and ExtendedTag

A tag is a 16 bit uint16_t. An Extended tag is a 32 bit uint32*t. It's really a pair of uint16*t. The extTag & 0xffff 16 == tag. The high bytes extTag & 0xffff0000 are the extension. It's usually 0x20000 which represents the root.

TiffVisitor State Tables and Functions

TiffCreator::tiffGroupStruct

The purpose of this table is to generate new objects to be pushed on the TiffVisitor stack.

```C++
/*
  This table describes the layout of each known TIFF group (including
  non-standard structures and IFDs only seen in RAW images).

  The key of the table consists of the first two attributes, (extended) tag
  and group. Tag is the TIFF tag or one of a few extended tags, group
  identifies the IFD or any other composite component.

  Each entry of the table defines for a particular tag and group combination
  the corresponding TIFF component create function.
 */
#define ignoreTiffComponent 0
const TiffGroupStruct TiffCreator::tiffGroupStruct_[] = {
    // ext. tag  group             create function
    //---------  ----------------- ---------------------------------------
    // Root directory
    { Tag::root, ifdIdNotSet,      newTiffDirectory<ifd0Id>                },

    // IFD0
    {    0x8769, ifd0Id,           newTiffSubIfd<exifId>                   },
```

TiffCreator::tiffTreeStruct

This is the state transition table.

```C++
/*
  This table lists for each group in a tree, its parent group and tag.
  Root identifies the root of a TIFF tree, as there is a need for multiple
  trees. Groups are the nodes of a TIFF tree. A group is an IFD or any
  other composite component.

  With this table, it is possible, for a given group (and tag) to find a
  path, i.e., a list of groups and tags, from the root to that group (tag).
*/
const TiffTreeStruct TiffCreator::tiffTreeStruct_[] = {
    // root      group              parent group      parent tag
    //---------  -----------------  ----------------- ----------
    { Tag::root, ifdIdNotSet,       ifdIdNotSet,      Tag::root },
    { Tag::root, ifd0Id,            ifdIdNotSet,      Tag::root },
```

This is a state table used to navigate the metadata heirachy. For example, starting at root, the first IFD wil create a new TiffDirectory and sets the state to ifd0Id. When tag 0x8769 is encountered, the parser will create new TiffDirectory and the state becomes exifId.

The "route" from the start of parsing (ifdIdNotSet), via the Tiff-EP tags (ifd0Id), via ExifTag/0x8769, to the MakerNote/0x927c, to the NikonPicture control is:

```C++
{ Tag::root, ifd0Id,            ifdIdNotSet,      Tag::root },
{ Tag::root, exifId,            ifd0Id,           0x8769    },
{ Tag::root, nikon3Id,          exifId,           0x927c    },
{ Tag::root, nikonPcId,         nikon3Id,         0x0023    },
```

When the state machine reaches nikonPcId, it manufactures a newTiffBinaryElement to decode it. This is a simple binary.

```C++
// Nikon3 picture control
{  Tag::all, nikonPcId,         newTiffBinaryElement        },
```

nikonPcCfg is defined as:

```C++
//! Nikon Picture Control binary array - configuration
extern const ArrayCfg nikonPcCfg = {
    nikonPcId,        // Group for the elements
    invalidByteOrder, // Use byte order from parent
    ttUndefined,      // Type for array entry
    notEncrypted,     // Not encrypted
    false,            // No size element
    true,             // Write all tags
    true,             // Concatenate gaps
    { 0, ttUnsignedByte,  1 }
};
//! Nikon Picture Control binary array - definition
extern const ArrayDef nikonPcDef[] = {
    {  0, ttUndefined,     4 }, // Version
    {  4, ttAsciiString,  20 },
    { 24, ttAsciiString,  20 },
...
```

```
    { 57, ttUnsignedByte,  1 }  // The array contains 58 bytes
};
```

The tags associated with nikonPcId are:

```C++
// Nikon3 Picture Control Tag Info
const TagInfo Nikon3MakerNote::tagInfoPc_[] = {
    TagInfo( 0, "Version", N_("Version"), N_("Version"), nikonPcId, makerTags, undefine
    TagInfo( 4, "Name", N_("Name"), N_("Name"), nikonPcId, makerTags, asciiString, 20, p
    TagInfo(24, "Base", N_("Base"), N_("Base"), nikonPcId, makerTags, asciiString, 20, p
...
};
```

This is a very flexible design. Not easy to understand. The design used by tvisitor.cpp is simpler 3.5 Presenting the data with visitTag().

TOC

7.9 Other Exiv2 Classes

The Metadatum and Key classes

The metadata consists of a key and data.

There are three derived classes of Metadatum: Exifdatum, Xmpdatum and Iptcdatum. These classes hold the actual data value. For example, in an Exifdatum, it contains the type/count/array from the Tiff Record. There are three derived classes of Key: ExifKey, XmpKey and IptcKey.

The MetaData Classes: ExifData, XmpData and IptcData

These are vector of the Key/Metadatum pairs. For example ExifData is std::vector. The following snippet from 7.2 Typical Sample Application](#7-2) shows how those are obtained and navigated:

```C++
Exiv2::Image::AutoPtr image = Exiv2::ImageFactory::open(path);
assert(image.get() != 0);
image->readMetadata();

Exiv2::ExifData &exifData = image->exifData();
if (exifData.empty()) {
    std::string error("No Exif data found in file");
    throw Exiv2::Error(Exiv2::kerErrorMessage, error);
}

for (Exiv2::ExifData::const_iterator i = exifData.begin(); i != exifData.end(); ++i) {
    std::cout << i->key() << " -> " << i->toString() << std::endl;
}
```

The Task Factory

This is implemented using ***Command*** in Design Patterns)

The Task Factory is used by the command-line utility exiv2 which supports sub-commands such as print, adjust, rename and extract. The TaskFactory returns an object with the Task Interface. The TaskFactory has to be created than then called to find the task runner. There is no equivalent in tvisitor.cpp.

The Image Factory

This is implemented using ***Factory*** in Design Patterns)

The purpose of the Image Factory is to create and BasicIo object and image handler and return this as an Image. Every image handler is required to define two global functions isImageFormatType() and newImageFormat(). For example: isJpegType() and newJpegInstance(). There is a table called the registry which defines the priority of image handlers and if isImageFormatType() returns true, his companion newImageFormat() is invoked to create the image object.

The implementation in tvisitor.cpp is simpler and requires no static functions and no static registry.

```C++
std::unique_ptr<Image> ImageFactory(std::string path)
{
    TiffImage tiff(path); if ( tiff.valid() ) return std::unique_ptr<Image> (new TiffIm
    JpegImage jpeg(path); if ( jpeg.valid() ) return std::unique_ptr<Image> (new JpegIm
    CrwImage  crw (path); if (  crw.valid() ) return std::unique_ptr<Image> (new  CrwIm
    PngImage  png (path); if (  png.valid() ) return std::unique_ptr<Image> (new  PngIm
    Jp2Image  jp2 (path); if (  jp2.valid() ) return std::unique_ptr<Image> (new  Jp2Im
    ICC       icc (path); if (  icc.valid() ) return std::unique_ptr<Image> (new  ICC
    PsdImage  psd (path); if (  psd.valid() ) return std::unique_ptr<Image> (new  PsdIm
    PgfImage  pgf (path); if (  pgf.valid() ) return std::unique_ptr<Image> (new  PgfIm
    MrwImage  mrw (path); if (  mrw.valid() ) return std::unique_ptr<Image> (new  MrwIm
    RiffImage riff(path); if ( riff.valid() ) return std::unique_ptr<Image> (new RiffIm
    RafImage  raf (path); if (  raf.valid() ) return std::unique_ptr<Image> (new  RafIm
    return NULL;
}
```

The Image Object and Image Parsers

I normally refer to the Image objects as Image Handlers. For example: JpegImage, TiffImage, PngImage. These are derived from the *abstract* Image class which offers an interface to the metadata engine. The most commonly used functions of the Image class are readMetadata() and writeMetadata(). There are various functions such as pixelWidth() and pixelWidth() which provide easy access properties of the image. Accessing this information in the MetaData is a little tedious:

```
Exiv2::ExifKey key("Exif.Photo.PixelXDimension");
if (exifData.findKey(key) != exifData.end()) {
    std::cout << exifData.findKey(key)->toString() << std::endl;
}
```

To simplify accessing Exif properties which could be defined in various location in the metadata, an easyaccess API is provide. This is described: 7.3 The EasyAccess API

The Image Parsers are required to provide both decode() and encode() methods which are called by image->readMetadata() and image->writeMetadata(). For example:

```C++
void OrfImage::readMetadata()
{
    if (io_->open() != 0) {
        throw Error(kerDataSourceOpenFailed, io_->path(), strError());
    }
    IoCloser closer(*io_);
    // Ensure that this is the correct image type
    if (!isOrfType(*io_, false)) {
        if (io_->error() || io_->eof()) throw Error(kerFailedToReadImageData);
        throw Error(kerNotAnImage, "ORF");
    }
    clearMetadata();
    ByteOrder bo = OrfParser::decode(exifData_,
                                     iptcData_,
                                     xmpData_,
                                     io_->mmap(),
                                     (uint32_t) io_->size());
    setByteOrder(bo);
} // OrfImage::readMetadata
```

TOC

8 Test Suite

Exiv2 has several different elements in the test suite. They are:

1. Bash Tests
2. Python Tests
3. Unit Test
4. Version Test

In writing this book, I want to avoid duplicating information from the Exiv2 documentation into this book. This book is intended to provide an engineering explanation of how the code works and why various design decisions were chosen. However, this book doesn't explain how to use Exiv2. How to use execute the test suite is documented in README.md.

TOC

8.1 Bash Tests

As the name implies, these tests were originally implemented as bash scripts. They started life as a collection of independant scripts which were written by different contributors. Although they all shared the goal of executing a command and comparing the output to a referenced file, there was no shared code. About 2012, I refactored the tests and put common code into test/functions.source. All bash tests begin by sourcing this file which performs environment checks, initialises bash variables and sets up bash functions such as copyTestFiles, runTest and reportTest.

Bash

```
#!/usr/bin/env bash
# Test driver for geotag

source ./functions.source

(   jpg=FurnaceCreekInn.jpg
    gpx=FurnaceCreekInn.gpx
    copyTestFiles $jpg $gpx

    echo --- show GPSInfo tags ---
    runTest                     exiv2 -pa --grep GPSInfo $jpg
    tags=$(runTest              exiv2 -Pk --grep GPSInfo $jpg  | tr -d '\r') # MSVC put
    echo --- deleting the GPSInfo tags
    for tag in $tags; do runTest exiv2 -M"del $tag" $jpg; done
    runTest                     exiv2 -pa --grep GPS     $jpg
    echo --- run geotag ---
    runTest                     geotag -ascii -tz -8:00 $jpg $gpx | cut -d' ' -f 2-
    echo --- show GPSInfo tags ---
    runTest                     exiv2 -pa --grep GPSInfo $jpg

) > $results 2>&1
reportTest

# That's all Folks!
##
```

The bash tests have been rewritten in python. This was done because running bash scripts on windows is painful for most Visual Studio users. The following script is a prototype in the project proposal to replace the bash scripts. The implementation in tests/bash_tests/utils.py is considerably more complicated as it emulates several system utilities including diff, md5sum, grep, xmllint and others. I am very grateful to Leo for the hard

work he performed to port bash_tests to python. Thank You, Leo.

The decision to convert bash scripts such as icc_test.sh to python was taken to achieve the following goals:

1. Cross Platform.
2. Simpler design than tests/system_tests.py.
3. Can be introduced as time permits.
4. No documentation changes!
5. We know the test is identical because we do not touch data/test.out.
6. Eliminate line-ending issues.
7. Eliminate diff, dos2unix, tr, pipes and other unix hackery.
8. Binary output support.

The project proposal is: https://github.com/Exiv2/exiv2/issues/1215

Here's the prototype in the proposal:

Python

```python
#!/usr/bin/env python3

import os
import shlex
import shutil
import subprocess

def error(s):
    print('**',s,'**')
def warn(s):
    print('--',s)

def chop(blob):
    lines=[]
    line=''
    for c in blob.decode('utf-8'):
        if c == '\n':
            lines=lines+[line]
            line=''
        elif c != '\r':
            line=line+str(c)
    if len(line) != 0:
        lines=lines+line
    return lines

def runTest(r,cmd):
    lines=[]
    try:
        # print('*runTest*',cmd)
        p        = subprocess.Popen( shlex.split(cmd), stdout=subprocess.PIPE,shell=Fals
        out,err  = p.communicate()
        lines=chop(out)
        if p.returncode != 0:
            warn('%s returncode = %d' % (cmd,p.returncode) )
    except:
        error('%s died' % cmd )

    return r,lines
```

```
    return r+lines

def echo(r,s):
    return r+[s]

def copyTestFiles(r,a,b):
    os.makedirs('tmp', exist_ok=True)
    shutil.copy('data/%s' % a,'tmp/%s' % a)
    shutil.copy('data/%s' % b,'tmp/%s' % b)
    return r

def cut(r,delim,field,lines):
    R=[]
    for line in lines:
        i = 0
        while i < len(line):
            if line[i]==delim:
                R=R+[line[i+1:]]
                i=len(line)
            else:
                i=i+1
    return r+R;

def reportTest(r,t):
    good = True
    R=chop(open('data/%s.out' % t ,'rb').read())
    if len(R) != len(r):
        error('output length mismatch Referance %d Test %d' % (len(R),len(r)))
        good=False
    else:
        i = 0
        while good and i < len(r):
            if R[i] != r[i]:
                error ('output mismatch at line %d' % i)
                error ('Reference: %s' % R[i])
                error ('Test:      %s' % r[i])
            else:
                i=i+1
    if not good:
        f=open('tmp/%s.out' % t , 'w')
        for line in r:
            f.write(line+'\n')
        f.close()

    print('passed %s' % t) if good else error('failed %s' %t )

# Update the environment
key="PATH"
if key in os.environ:
    os.environ[key] = os.path.abspath(os.path.join(os.getcwd(),'../build/bin'))
                    + os.pathsep + os.environ[key]
else:
    os.environ[key] = os.path.abspath(os.path.join(os.getcwd(),'../build/bin'))

for key in [ "LD_LIBRARY_PATH", "DYLD_LIBRARY_PATH" ]:
    if key in os.environ:
        os.environ[key] = os.path.abspath(os.path.join(os.getcwd(),'../build/lib'))
```

```
95                          + os.pathsep + os.environ[key]
96      else:
97          os.environ[key] = os.path.abspath(os.path.join(os.getcwd(),'../build/lib'))
98
99  r=[]
100 t=  'geotag-test'
101
102 warn('%s'       % t)
103 warn('pwd=%s'   % os.getcwd())
104 warn('exiv2=%s' % shutil.which("exiv2"))
105
106 jpg='FurnaceCreekInn.jpg'
107 gpx='FurnaceCreekInn.gpx'
108
109 r=          copyTestFiles(r,jpg,gpx)
110 r=          echo   (r,'--- show GPSInfo tags ---')
111 r=          runTest(r,'exiv2 -pa --grep GPSInfo tmp/%s' % jpg)
112
113 r=          echo   (r,'--- deleting the GPSInfo tags')
114 tags=       runTest([],'exiv2 -Pk --grep GPSInfo tmp/%s' % jpg)
115 for tag in tags:
116      r=     runTest(r,'exiv2 -M"del %s" tmp/%s' % (tag,jpg))
117 r=          runTest(r,'exiv2 -pa --grep GPS  tmp/%s' %  jpg)
118 r=          echo   (r,'--- run geotag ---')
119 lines=      runTest([],'geotag -ascii -tz -8:00 tmp/%s tmp/%s' % (jpg,gpx))
120 r=          cut    (r,' ',2,lines)
121 r=          echo   (r,'--- show GPSInfo tags ---')
122 r=          runTest(r,'exiv2 -pa --grep GPSInfo tmp/%s' % jpg)
123
124 reportTest(r,t)
125
126 # That's all Folks
127 ##
```

TOC

8.2 Python Tests

Before we proceed to discuss the python tests, I want to thank Dan for his work on this. His framework has been in service for more than 2 years without issue. The code is robust and flexible. In addition to inventing the framework, Dan also converted hundreds of bash scripts into python scripts. Thank You, Dan for doing such a wonderful job.

Here is a typical python test. See https://github.com/Exiv2/exiv2/pull/992. The user provided a test python file tests/bugfixes/github/test_pr_992.py

```python
# -*- coding: utf-8 -*-

import system_tests

class NikonSigmaLens_APO_MACRO_180_F35_EX_DG_HSM(metaclass=system_tests.CaseMeta):
    url = "https://github.com/Exiv2/exiv2/pull/992"

    filename = "$data_path/Sigma_APO_MACRO_180_F3.5_EX_DG_HSM.exv"
    commands = ["$exiv2 -pa --grep lensid/i $filename"]
    stderr = [""]
    stdout = [""
        """Exif.NikonLd3.LensIDNumber                   Byte        1  Sigma APO Macro 1
"""
]
    retval = [0]
```

The test file is test/data/Sigma_APO_MACRO_180_F3.5_EX_DG_HSM.exv. The tests executes the program `exiv2 -pa --grep lensid/i foo.exv` and compares the output to stdout. That's it.

In this case, there is only one exiv2 command being executed on a single file. Most tests are more involved. You'll notice that the variables, commands, stderr, stdout and retval are arrays to make it easy to execute several commands in a single test. The most common program to run is exiv2, however other programs from build/bin or system commands can be invoked. You may not pipe data between commands, however you can redefine stdin/stdout filter decode_output if required.

The system_test class provides useful decorators including class CopyFiles, CopyTmpFiles and DeleteFiles to manage files.

There are two test suites:

1. test/ are written in bash test/*.sh are the bash scripts
   test/data are reference files
   test/tmp is used to store script output (for comparison to reference output)

2. tests/ are written in python the tests are run with the command `python3 runner.py`
   tests/bugfixes/ has python scripts containing code and reference output
   tests/bash_tests has python scripts which will replace test/*.sh in Exiv2 v0.27.4 and v0.28

```
541 rmills@rmillsmbp:~/gnu/github/exiv2/0.27-maintenance/tests $ python3 runner.py -h
usage: runner.py [-h] [--config_file CONFIG_FILE] [--verbose] [--debug] [dir_or_file]

The system test suite

positional arguments:
  dir_or_file           root directory under which the testsuite searches for tests or (
                        file'slocation)

optional arguments:
  -h, --help            show this help message and exit
  --config_file CONFIG_FILE
                        Path to the suite's configuration file
  --verbose, -v         verbosity level
  --debug               enable debugging output
542 rmills@rmillsmbp:~/gnu/github/exiv2/0.27-maintenance/tests $
```

Bash

When you add a test, you should choose a class for your test that is unique. You can debug it with Visual Studio Code (and probably other python debuggers). Test it from the terminal with:

```
$ cd tests
$ python3 runner.py --verbose bugfixes/github/sigma_18_35_DC_HSM.py
```

When you're confident that it's working, run the complete test suite.

```
$ cd build
$ make tests  # VERBOSE=1 will create more output
```

Bash

The command: `$ make tests` executes the command `$ cd tests ; python3 runner.py` which recursively searches bugfixes and executes every python file.

TOC

8.3 Unit Test

The unit tests are very useful for testing C++ functions with a well defined input and output. In Chemistry, we have elements and compounds. The unit tests are good for testing elements of the software. The unit tests are written in C++ and use the Google Test library. Here's a typical test progam, extracted from unitTests/test_futils.cpp

```C++
#include <exiv2/exiv2.hpp>
#include <exiv2/futils.hpp>

// Auxiliary headers
#include <fstream>
#include <cstdio>
#include <cerrno>
#include <stdexcept>

#include "gtestwrapper.h"

using namespace Exiv2;

TEST(base64decode, decodesValidString)
{
    const std::string original ("VGhpcyBpcyBhIHVuaXQgdGVzdA==");
    const std::string expected ("This is a unit test");
    char * result = new char [original.size()];
    ASSERT_EQ(static_cast<long>(expected.size()+1),
              base64decode(original.c_str(), result, original.size()));
    ASSERT_STREQ(expected.c_str(), result);
    delete [] result;
}
```

The output is:

```
[----------] 1 test from base64decode
[ RUN      ] base64decode.decodesValidString
[       OK ] base64decode.decodesValidString (0 ms)
[----------] 1 test from base64decode (0 ms total)
```

To build and execute unit tests:

```Bash
$ cmake .. -DEXIV2_BUILD_UNIT_TESTS=1
$ cmake --build . --config Release --target unit_tests # or make unit_tests
$ cmake --build . --config Release --target unit_test  # or make unit_tests
```

The unit tests are built into a single executable bin/unit_tests(.exe)

TOC

8.4 Version Test

The version test is *more-or-less* the output of the command *$ exiv2 –verbose –version* which produces about 150 lines of output. About 60 lines of the output are a list of pre-registed XMP namespaces and of little interest. So, the script test/version-test.sh counts and filters out the XMP namespaces.

```bash
#!/usr/bin/env bash
# Test driver for exiv2.exe --verbose --version

source ./functions.source

(   cd "$testdir"
    # Curiously the pipe into grep causes FreeBSD to core dump!
    if [ $(uname) != "FreeBSD" ]; then
                                    runTest exiv2 --verbose --version | grep -v ^xmlns
        echo xmlns entry count: $(runTest exiv2 --verbose --version | grep    ^xmlns | w
    else
        runTest exiv2 --verbose --version
    fi
)

# That's all Folks!
##
```

The implementation of the command *$ exiv2 –verbose –version* and the version number scheme is discussed in detail: 11.9 Releases.

8.5 Generating HUGE images

Before getting into a discussion about this, I'd like to thank several collaborators who have contributed to this part of the book. Joris Van Damme of AWare Systems maintains the BigTiff web-site and was very helpful on email. This topic was also discussed at: https://github.com/Exiv2/exiv2/issues/1248 and I wish to thank LeoHsiao1 and kolt54321 for their input.

Almost all the test images in the Exiv2 test have been added in response to bugs ***or*** during feature development. Most of the test files are checked into the repository. There is an svn directory with larger test images which are downloaded on demand by the test suite. svn://dev.exiv2.org/svn/team/test

As Exiv2 moved from 32 to 64 bits, the size of images has grown. HUGE files > 3GB are commonly used in medical and space imaging applications. For years, I've wanted to undertake a project to test if Exiv2 can really handle HUGE files. The obvious way to work with them is to generate them on demand.

I've looked at several libraries for the purpose of generating HUGE files.

1. libtiff-4 (which supports BigTiff)
2. PIL (python imaging library)
3. FreeImage is a wrapper for libJPEG,libtiff, libpng, LibOpenJPEG, LibJXR, LibRAW, LibWebP and OpenEXR.

10.5.1 libtiff-4 with BigTiff support

1. Build and install jpeg-6b:

```Bash
$ cd ~/gnu/jpeg/jpeg-6b
$ make clean ; ./configure --prefix=/usr/local ; make
$ sudo make install
$ sudo cp j*.h  /usr/local/include
```

2 Build and install zlib:

```Bash
$ ~/gnu/zlib/zlib-1.2.11
$ ./configure --prefix=/usr/local
$ make
$ sudo make install
```

3 Download and install libtiff4 from http://download.osgeo.org/libtiff/

```Bash
$ cd ~/gnu/tiff/libtiff-4.1.1
$ cd build
$ cmake ..
$ sudo make install
```

4 Build the program create_tiff from the book resources:

```Bash
$ g++ create_tiff.cpp -ltiff -lz -o create_tiff
$ ./create_tiff
usage ./create_tiff width height path [open-option]
```

You may wish to use the code in CMakeLists.txt to build create_tiff. However most folks have neither *-ljpeg*

nor *-ltiff* installed, so the default CMakeList.txt does not build create_tiff.

5 Testing BigTiff

The parameter open-option defaults to w8 which means "write a BigTiff Little Endian". Other options useful are "w4", "w8b" and "w4b". The documentation for this is at:
file:///usr/local/share/doc/tiff/html/man/TIFFOpen.3tiff.html

```Bash
.../book $ ./create_tiff 200 400 foo.tif ; ls -l foo.tif ; build/tvisitor foo.tif
TIFFScanlineSize64: Computed scanline size is zero.
-rw-r--r--+ 1 rmills  staff  320252 16 Jul 18:39 foo.tif
STRUCTURE OF BIGTIFF FILE (II): foo.tif
.../book $ ./create_tiff 200 400 foo.tif w4 ; ls -l foo.tif ; build/tvisitor foo.tif | h
TIFFScanlineSize64: Computed scanline size is zero.
-rw-r--r--+ 1 rmills  staff  320154 16 Jul 18:40 foo.tif
STRUCTURE OF TIFF FILE (II): foo.tif
.../book $ ./create_tiff 200 400 foo.tif wb4 ; ls -l foo.tif ; build/tvisitor foo.tif |
TIFFScanlineSize64: Computed scanline size is zero.
-rw-r--r--+ 1 rmills  staff  320154 16 Jul 18:40 foo.tif
STRUCTURE OF TIFF FILE (MM): foo.tif
.../book $ ./create_tiff 200 400 foo.tif wb8 ; ls -l foo.tif ; build/tvisitor foo.tif |
TIFFScanlineSize64: Computed scanline size is zero.
-rw-r--r--+ 1 rmills  staff  320252 16 Jul 18:40 foo.tif
STRUCTURE OF BIGTIFF FILE (MM): foo.tif
```

You can create HUGE BigTiff files:

```
.../book $ ./create_tiff 80000 20000 foo.tif  ; ls -lh foo.tif ; build/tvisitor foo.tif
TIFFScanlineSize64: Computed scanline size is zero.
-rw-r--r--+ 1 rmills  staff   6.0G 16 Jul 18:43 foo.tif
STRUCTURE OF BIGTIFF FILE (II): foo.tif
 address |    tag                              |      type |    count |    offset | valu
2105032728 | 0x0100 Exif.Image.ImageWidth        |      LONG |        1 |           | 80
2105032748 | 0x0101 Exif.Image.ImageLength       |     SHORT |        1 |           | 20
2105032768 | 0x0102 Exif.Image.BitsPerSample     |     SHORT |        4 |           | 8
2105032788 | 0x0103 Exif.Image.Compression       |     SHORT |        1 |           | 1
2105032808 | 0x0106 Exif.Image.PhotometricInte.. |     SHORT |        1 |           | 2
2105032828 | 0x0111 Exif.Image.StripOffsets      |     LONG8 |        1 |           | 16
2105032848 | 0x0112 Exif.Image.Orientation       |     SHORT |        1 |           | 1
2105032868 | 0x0115 Exif.Image.SamplesPerPixel   |     SHORT |        1 |           | 4
2105032888 | 0x0116 Exif.Image.RowsPerStrip      |      LONG |        1 |           | 32
2105032908 | 0x0117 Exif.Image.StripByteCounts   |     LONG8 |        1 |           | 64
2105032928 | 0x011c Exif.Image.PlanarConfigura.. |     SHORT |        1 |           | 1
END: foo.tif
.../book $
```

I have not investigated the message *TIFFScanlineSize64: Computed scanline size is zero.*

10.5.1 PIL (python imaging library)

PIL is the Python Imaging Library. The clone Pillow is well maintained and documented. It's very impressive. Interestingly, PIL has some metadata capability to deal with Exif, XMP, ICC and IPTC data.

For the moment, I've started a little python program to create images. I'll add options to this. At the moment, you use it like this:

```Bash
$ width=50000 ; height=50000 ; path=foo.png ; ./create_image.py $width $height $pa
-rw-r--r--+ 1 rmills  staff   7.0G 17 Jul 14:06 foo.png
STRUCTURE OF PNG FILE (MM): foo.png
$
```

Both exiv2 and tvisitor parse this file in 0.2 seconds and say "No metadata".

```Python
#!/usr/bin/env python3

import sys
from   PIL import Image

##
#
def main(argv):
    """main - main program of course"""

    argc     = len(argv)
    if argc < 2:
        syntax()
        return

    width  = int(argv[1])
    height = int(argv[2])
    path   =     argv[3]

    img    = Image.new('RGB', (width, height), color = 'red')
    img.save(path,compress_level=0)

##
#
if __name__ == '__main__':
    main(sys.argv)
```

10.5.3 FreeImage

This library is a wrapper for several open source graphics libraries and can generate PNG, JPEG and other formats. Regrettably, it appears to be no longer supported and has not been updated since 2018.

The code is available here: https://freeimage.sourceforge.io/download.html

I have successfully built this with Visual Studio and on Linux. I couldn't get it to build on macOS, although I think it will build with a little more effort.

I've found API documentation here: https://mirrors.dotsrc.org/exherbo/FreeImage3170.pdf

I've found a user guide here: http://graphics.stanford.edu/courses/cs148-10-summer/docs/UsingFreeImage.pdf

I wrote an example program:

```C++
// g++ example.cpp -o example -lfreeimage -L.
#include <iostream>
#include <FreeImage.h>
#define   WIDTH 600
#define  HEIGHT 800
#define     BPP  24

using namespace std ;

int main(int argc,const char* argv[])
{
    FreeImage_Initialise();

    FIBITMAP* bitmap = FreeImage_Allocate(WIDTH, HEIGHT, BPP);
    if (bitmap) {
      for (int i=0; i<WIDTH; i++) {
        for (int j=0; j<HEIGHT; j++) {
            RGBQUAD color ;
            color.rgbRed   = 0;
            color.rgbGreen = (double) i /  WIDTH * 255.0 ;
            color.rgbBlue  = (double) j / HEIGHT * 255.0 ;
            FreeImage_SetPixelColor(bitmap,i,j,&color);
      } }
      const char* image = "example.png";
      if (FreeImage_Save(FIF_PNG, bitmap, image, 0)) {
         cout << "Image successfully saved: " << image << endl;
      } else {
         cerr << "Unable to save image!" << endl;
      }
    } else {
      cerr << "Unable to create bitmap!" << endl;
    }

    FreeImage_DeInitialise (); //Cleanup!
}
```

The largest file I produced with freeimage was 1.8gb. I suspect the framebuffer is limited to 32bits (3.2gb). FreeImage has metadata support to read/write metadata blocks and possibly list key/value pairs.

TOC

9 API/ABI Compatibility

This is discussed: https://github.com/Exiv2/exiv2/issues/890

I believe there are tools to help with this, however I haven't successfully used them. Let's define a couple of terms:

Acronym	Meaning	Description
API	Application Program Interface	How exiv2 appears to its user. Defined in include/exiv2/exiv2.hpp
ABI	Application Binary Interface	What libexiv2 requires from the system

Tools to reveal API and ABI

To reveal the API, list all the entry points defined in the library:

```Bash
00000000002576c0 T _WXMPIterator_DecrementRefCount_1
0000000000257610 T _WXMPIterator_IncrementRefCount_1
0000000000257800 T _WXMPIterator_Next_1
...
000000000002a8f0 T std::__1::basic_istream<char, std::__1::char_traits<char> >&...
000000000008ccd0 T _ini_parse
000000000008cc80 T _ini_parse_file
000000000008c610 T _ini_parse_stream
```

To reveal the ABI, list all the unsatisfied entry points:

```Bash
                 U _XML_Parse
                 U _XML_ParserCreateNS
...
                 U _time
                 U _uncompress
                 U _vsnprintf
                 U dyld_stub_binder
```

To reveal the libraries to be dynamically loaded (on macOS):

```Bash
726 rmills@rmillsmbp:~/gnu/github/exiv2/0.27-maintenance/build $ otool -L lib/libe
lib/libexiv2.dylib:
    @rpath/libexiv2.27.dylib (compatibility version 27.0.0, current version 0.27.3)
    /usr/local/lib/libz.1.dylib (compatibility version 1.0.0, current version 1.2.11)
    /usr/local/lib/libintl.8.dylib (compatibility version 10.0.0, current version 10.2.0
    @rpath/libexpat.1.dylib (compatibility version 1.0.0, current version 1.6.11)
    /usr/lib/libiconv.2.dylib (compatibility version 7.0.0, current version 7.0.0)
    /usr/lib/libc++.1.dylib (compatibility version 1.0.0, current version 902.1.0)
    /usr/lib/libSystem.B.dylib (compatibility version 1.0.0, current version 1281.100.1)
727 rmills@rmillsmbp:~/gnu/github/exiv2/0.27-maintenance/build $
```

Shared object dependencies on Linux/Unix/Cygwin/MinGW can be inspected with ldd. Visual Studio Users can use dumpbin.exe.

```Bash
34 rmills@ubuntu:~/gnu/github/exiv2/0.27-maintenance/build $ ldd lib/libexiv2.so
    linux-vdso.so.1 (0x00007ffca47e9000)
    libz.so.1 => /lib/x86_64-linux-gnu/libz.so.1 (0x00007f0efd778000)
    libexpat.so.1 => /lib/x86_64-linux-gnu/libexpat.so.1 (0x00007f0efd74a000)
    libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007f0efd727000)
    libstdc++.so.6 => /lib/x86_64-linux-gnu/libstdc++.so.6 (0x00007f0efd546000)
    libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x00007f0efd3f7000)
    libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f0efd205000)
    libgcc_s.so.1 => /lib/x86_64-linux-gnu/libgcc_s.so.1 (0x00007f0efd1e8000)
    /lib64/ld-linux-x86-64.so.2 (0x00007f0efdb99000)
35 rmills@ubuntu:~/gnu/github/exiv2/0.27-maintenance/build $
```

To discover which libraries are loaded at run-time:

```Bash
727 rmills@rmillsmbp:~/gnu/github/exiv2/0.27-maintenance/build $ bin/exiv2 -vVg li
exiv2 0.27.3
library=/Users/rmills/gnu/github/exiv2/0.27-maintenance/build/lib/libexiv2.0.27.3.dylib
library=/usr/local/lib/libintl.8.dylib
library=/usr/lib/libc++.1.dylib
library=/usr/lib/libSystem.B.dylib
library=/usr/local/lib/libz.1.dylib
library=/usr/local/lib/libexpat.1.6.11.dylib
...
library=/usr/lib/libicucore.A.dylib
library=/usr/lib/libz.1.dylib
728 rmills@rmillsmbp:~/gnu/github/exiv2/0.27-maintenance/build $
```

Changing the API

This should be done with great caution. If an application requires an entry point that is not defined, it will usually refuse to launch the application. You should therefore never remove an entry point from a library. Changing the signature of an API is effectively to remove an entry point and introduced a new entry point.

If a library offers an entry point which is not used by an application, the library will be loaded and the application will launch.

So, the rules are:

1. Never remove an entry point or data structure.
2. Never change the signature of an entry point or data structure.
3. It is OK to add new entry points and data structures.

Caution: When you add a new entry point or data structure, applications compiled with the new library will be unable to "downgrade" to an earlier version of the library. Best practice is to never change the API.

Testing for DLL compatibility

For Exiv2 v0.27 “dots”, I:

1. Build v0.27 and test
2. Build v0.27.X and test
3. Over-write the v0.27 library with the v0.27.X library and test v0.27

There will of course be test exceptions, however the test suite should run without crashing.
https://github.com/Exiv2/exiv2/issues/890#issuecomment-613611192

TOC

10 Security

10.1 Security Policy

GitHub provides a “Security” tab in the User Interface. You can define a file SECURITY.md to define your security policy. https://github.com/Exiv2/exiv2/security/policy There are other tabs below security which deal with Notifications of different kinds. I don’t understand most of the GitHub Security Machinery.

Security alerts are published here: https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=Exiv2 We open an issue with the label “Security” on GitHub and fix it. It doesn’t get special treatment and the fix will be included in the next release of the branch.

Exiv2 does not back-port security (or any other fix) to earlier releases of the code. An engineer at SUSE has patched and fixed some security releases for Exiv2 v0.26 and Exiv2 v0.25 in branches 0.26 and 0.25.

The Exiv2 “dot” releases such as v0.27.2 include security fixes, bug fixes and minor feature and documentation updates. Exiv2 has never issued a “security release” which would be an existing release PLUS one *or more* security PRs. The version numbering scheme is explained here: 11.9 Releases. The design includes provision for a security release.

I was very impressed by the libssh security process which has provision to issue security notices to third parties. Exiv2 is not sufficiently resourced to support this capability. If the community decide that Exiv2 must strengthen its security process, the community will have to provide the necessary resources.

TOC

10.2 The Fuzzing Police

We received our first CVE from the fuzzing police in July 2017. Not a pleasant experience. It was delivered in a blog post demanding that we re-write Exiv2 as it was “unsafe”. Needless to say, no resources were being offered for the re-write, no justification was offered and no explanation why a re-write of 100,000 lines of code would fix anything.

A couple of years later, Kevin send us four security alerts. When I invited him to solve them, he agreed. He subsequently wrote this interesting and helpful article.

https://securitylab.github.com/research/how-to-escape-from-the-fuzz

While security is an important matter, the behaviour of the fuzzing police is despicable and very demotivating. They frequently report false positives which consume/waste resources. None of those people ever say “Thank You” when something is fixed and never apologise for false positives. They sometimes say something useless like “I did you a favour because there could have been something wrong.”.

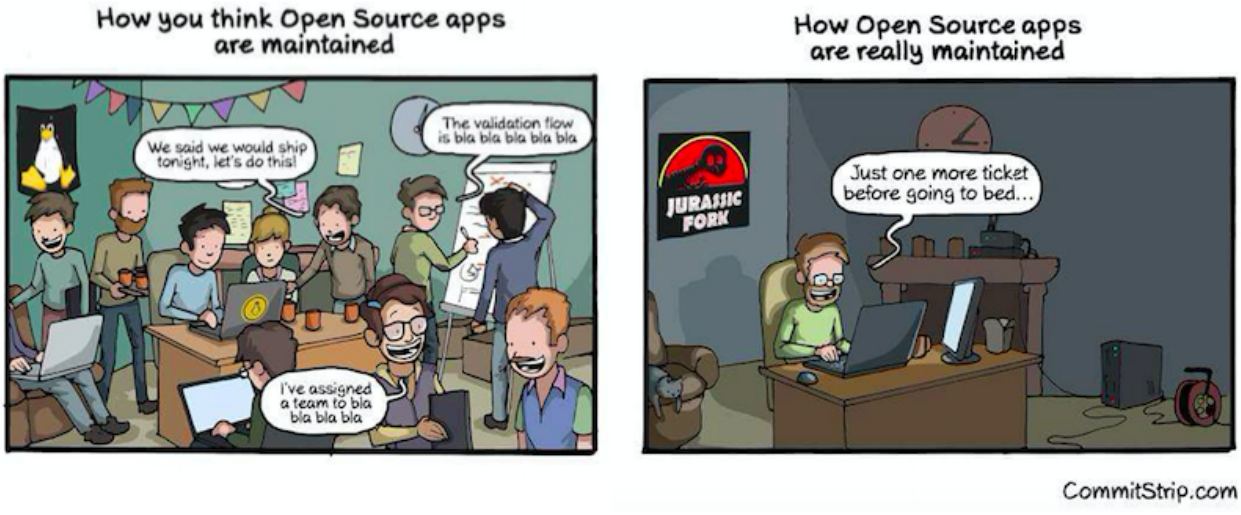
I must also mention that the fuzzing police use special tools that build and instrument the code to detect suspicious run-time activity. Often, there is no end-user bug report to demonstrate an issue. When they report an issue, they provide a file called `poc` = Proof of Concept. Their bug reports are usually totally devoid of information about how to reproduce the issue and there is no cross-reference with the CVE tracking data-base.

Everything is treated as urgent. All their reports are assigned very high levels of vulnerability. In short, those people are a pain in the butt and waste enormous amounts of Team Exiv2 engineering resources.

As the fuzzing police maintain their own CVE data base, the number and frequency of security issues is tracked and published. Their mission in life is negative. I don’t have a good word to say about those peple.

TOC

11 Project Management

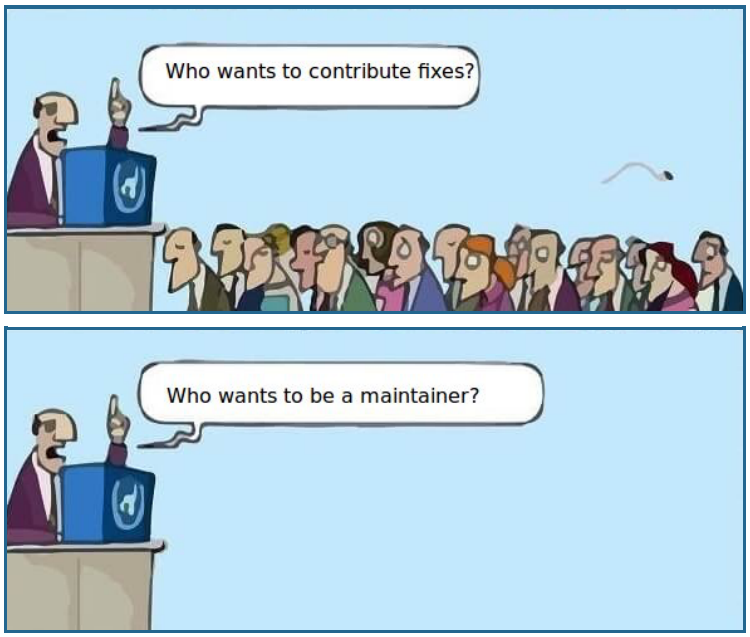


This topic deserves a book in its own right. It's easy to think of an Open Source Project as source code. It's not. The source code is a major part of the project, however probably only 50% of the effort goes into the code. We have many stakeholders in a project including: contributors, users, security, distros, and competitors. The project needs documentation, build, test, bug reporting and many other elements.

You may have seen the sketch in ***The Life of Brian*** which begins with John Cleese asking the question ***"What have the Romans Ever Done for Us?"*** and somebody replies ***The Aqueduct***. Within one minute they list Water Supply, Roads, Schools, Sanitation, Police, Laws, Medicine, Wine and Public Health. It's much the same with Open Source. Of course we have source code, however we also have Build, Test, Platform Support, Documentation, User Support, Security, Release Engineering, Localisation and other matters that require time and attention.

You are probably not surprised to learn that most stakeholders consider their concern should be the top priority for the project. The challenge is that there are many stakeholders and therefore many top priorities. When dealing with a stakeholder's issue, they frequently say ***All you have to do is bla bla bla***. In my head, I hear the words in a slightly different order. I hear ***You have to do it all***.

The difficulties of maintaining an open-source project are well explained in this article: https://steemit.com/opensource/@crell/open-source-is-awful from which I have copied this cartoon:



I will quote the following as it seems totally true.

If most businesses are using Open Source code for free, how are the developers compensated for that real time and effort? In the majority of cases the answer is ***with verbal abuse and emotional blackmail****.*

The very large projects (the Linux kernel, the Firefox web browser, etc.) end up with a few smart companies realizing it's in their self-interest to fund full time development, and most of their work ends up being non-volunteer. They're not the problem. The problem is the mid-range to small projects, maintained by volunteers, that get short-shrifted. People don't value free.

Not a month goes by without some maintainer of an Open Source project throwing up their hands in frustration and walking away because of burnout. Burnout caused invariably by the demands that people make of their free time. The code was free, so why isn't free support and personalised help available for life???

I am astonished at the verbal abuse I have received. About every three years I receive an email from somebody I have never met thanking me for my efforts. I get daily emails of criticism and complaint. I will not name a French Engineer on whose behalf I have spent hundreds of hours. Not once has he expressed appreciation. His emails and public posts of criticism are brutal.

When somebody provides a patch, they seldom provide test code, update the documentation or modify the build scripts. The feature is often incomplete. For example, in adding a new platform, nobody has ever provided platform specific code in src/version.cpp and src/futils.cpp. Sometimes they break all the sample applications. When I ask them to finish the job, they say: "oh you can do that.". Nobody ever maintains or supports their patch. Contributors frequently refuse to modify a patch when asked to do so in a review.

I have found recruiting contributors to be very challenging and difficult. I appreciate the work done by everybody who has contributed. The future of Exiv2 is a matter for the community. Perhaps this book will inspire somebody to maintain Exiv2 or write a replacement.

How to be an AB

I'll leave you to figure out what an **AB** is. It might be ***Annoying Bikeshedder***. They come in different versions.

The first is a **DAB** who deliberately blocks progress. Before we used GitHub, the only way to release a new version of Exiv2 was via the web-site. Sure we could tag a version in SVN, however this was passive. I'm not aware of any central organisation which tracks the versions of projects. So, publishing requires the password to enable the site to be updated. And there's only one person who knows the password. There are other versions of this kind of obstruction. For example, you can own the domain registration or hosting contract and allow them to expire.

I think the need for a web-site for the project has been mostly replaced by GitHub. We can publish new releases (and pre-releases). However the effort to transfer all project resources to GitHub is considerable. We have had complaints about the repository being too big, so we have an SVN repository for team resources such as old releases, this book, the project logo, minutes of team meetings and so on. And while I understand the team's hostility to SVN, no sensible alternative has been proposed.

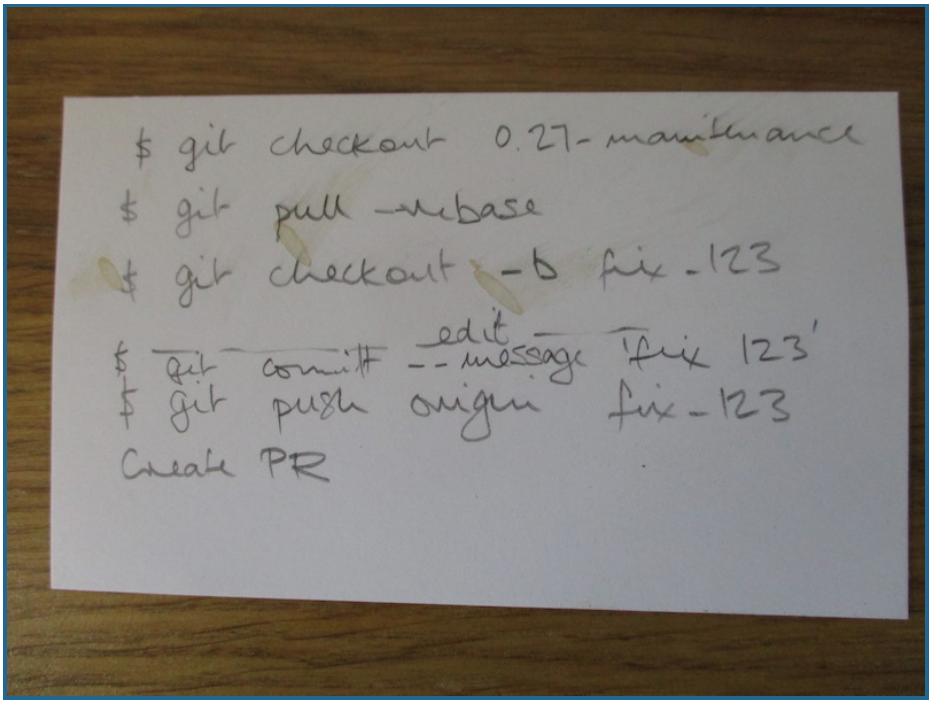
The first couple of releases I published on GitHub were not instantly tagged. Guess what? Within hours, somebody complained. People have complained about the version numbering scheme. GitHub automatically generates bundles when you create a release. Somebody complained about them. Somebody complained that the pre-release web-site was too similar to the release web-site (although every page is labelled) so I put "Pre-Release" on the background of every page. You might expect that complaints would include words of appreciation for the effort to make the release. You'd be wrong. Complaints are normally abrupt. Words such as ***Please*** and ***Thank You*** are seldom used by members of the community.

On the subject of web-sites, I must mention DOS attacks. DOS is Denial of Service. Occasionally, exiv2.org will deliver a message about being severely busy. This is because the web-site is being subjected to about 100 HTTP get requests per second. Every request is from a different IP address. These attacks can continue for several

days. Why does somebody attack exiv2.org in this way? How can you defend exiv2.org from such an attack? So, we'll call that **MAB** for Malicious.

How about **TAB** which is to change the project tools. Git came close to killing me. I know many people love Git and think it's the greatest thing ever invented. I don't. I worked on Acrobat at Adobe. A big project. When I retired in 2014, there were about 200 developers who had been working for 20 years on 25 million lines of code. To build it, you need 100GBytes of free space. How can git handle such a monster when every repos has 100% of the code and 100% of the project history? Nobody has given me an answer.

When we adopted Git, it took me 2 years to figure out how to submit a PR. I purchased the book **Pro Git**. It doesn't cover PRs. So, the only way to submit code is undocumented. I am very grateful to Luis and Andreas S for helping me with Git. I eventually wrote this on a card:



The funny brown marks were added by our cat Lizzie. I'd just written this card when she arrived in my office fresh from a hunt in the garden. I don't know what she did to the card. She's never done anything like this before or since. She expressed her opinion of git.

Another flavour is the **AAB** which I reserve for the fuzzing police. The **A** stands for **Aggressive**. I've discussed my dislike of these people here: 10.2 The Fuzzing Police

Or there's the review mechanism which I'll dub **RAB** You insist:

1. All code changes must be approved.
2. No contributor can approve their own change.
3. Nobody reviews or approves any code change.

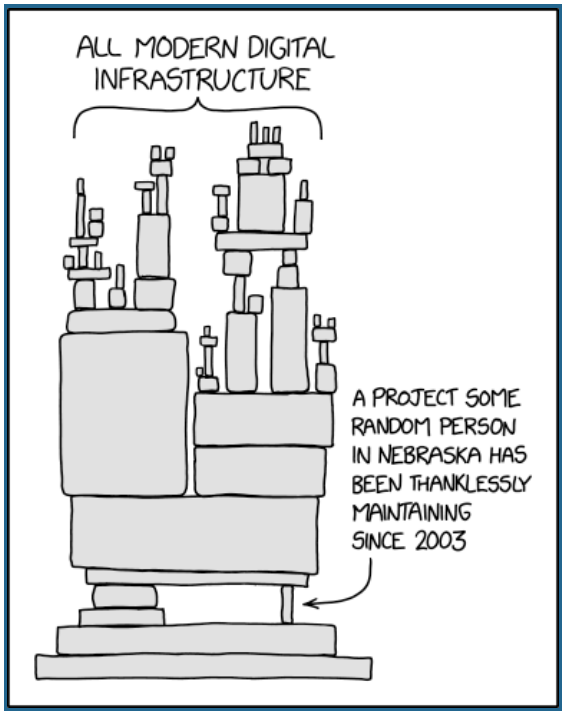
There are many other forms of **AB**. For example, there is legal **LAB**. This involves a legal challenge. You say "We might be infringing somebody's patent!". This is particularly effective as you don't need to provide evidence. Even if there is a written legal opinion you can refute that with the words: "The legal opinion has not been tested in court.". The case has not been tested in court for the obvious reason that it is not illegal. This

show-stopper was used by two contributors to block ISOBMFF support in Exiv2 v0.27.3. I received more than 100 emails from users asking "What is the legal problem?", so I called a meeting on Zoom and users on 5 continents attended. The two contributors who raised the show-stopper did not bother to turn up, although one emailed later to say "Apologies. I fell asleep on the couch and missed the meeting.".

How about this method? You complain about a font being used. We'll call this **FAB**. This is very effective because you're only asking for a 100% reformat of the book and all the graphics. That's not much to ask. When I designed the Exiv2 Logo, a contributor asked for the font to be changed. I proposed alternatives and received no response.

As for myself, I am a **CAB** where *C* stands for *clever*. However I am an **AB** and that's why I've found it difficult to recruit and retain contributors.

There are so many ways to incur the outrage of stakeholders. And so many ways in which people can and do complain. All in all, working on an open-source project is a thank-less task. When I released v0.25, a contributor in Peru said on Facebook ***Robin should get a medal for his work. Exiv2 would have died years ago without his commitment.*** So I asked my family to write to the UK Government to propose that I be given an honour. The family silently refused. Alison comforted me by saying ***Nobody is ever going to thank you for working on Exiv2.***



Solutions to the issue of ABs

There are ways to fix on-line abuse. We could do what the Fuzzing Police do. They do not negotiate with any project. They arrive unexpectedly and deliver their message. And they track your response and performance.

The Community would be well served by a similar task force to investigate complaints about the behaviour of stakeholders and maintainers. There are undoubtably stakeholders who would report me. However I would be able to complain about a stakeholder if I felt they had behaved inappropriately.

For sure, I would not welcome my performance being reviewed. However a fair, honest and independant review of an issue would be helpful. The stakeholder and I would shake hands and learn from the situation. If the stakeholder or maintainer do not attend the review, their github account should be suspended.

Another possible solution would be to register a complaint about an individual. In the same way as a bug report can be opened on a project, it would be helpful to open a complaint about an individual. And that complaint can only be closed by the person who opened it. I'm fairly sure, we'd soon discover two things:

1. A few people raise many complaints about other people.
2. There are many open complaints about a few people.

While there is no sanction for a stakeholder being unreasonable, bad behaviour will continue. A solution is possible. If you are thinking ***I never contribute, therefore this does not affect me!***, I ask you to think again if you use open-source. When maintainers are abused and leave their project, everybody looses out.

Will the Community do something about on-line abuse?

TOC

11.1 C++ Code

Exiv2 is written in C++. Prior to v0.28, the code was written to the C++ 1998 Standard and makes considerable use of STL containers such as vector, map, set, string and many others. The code started life as a 32-bit library on Unix and today builds on 32 and 64 bit systems running Linux, Unix, macOS and Windows (Cygwin, MinGW, and 7 editions of Visual Studio). It can be build by GCC or Clang. Although the Exiv2 project has never supported Mobile Platforms or Embedded Systems, it should be possible to build for other platforms with modest effort.

The code has taken a great deal of inspiration from the book Design Patterns: Elements of Reusable Object=Oriented Software.

Starting with Exiv2 v0.28, the code requires a C++11 Compiler. Exiv2 v0.28 is a major refactoring of the code and provides a new API. The project maintains a series of v0.27 "dot" releases for security updates. These releases are intended to ease the transition of existing applications in adapting to the new v0.28 API.

TOC

11.2 Build

The build code in Exiv2 is implemented using CMake: cross platform make. This system enables the code to be built on many different platforms in a consistant manner. CMake recursively reads the files CMakeLists.txt in the source tree and generates build environments for different build systems. For Exiv2, we actively support using CMake to build on Unix type plaforms (Linux, macOS, Cygwin, MinGW, NetBSD, Solaris and FreeBSD), and several editions of Visual Studio. CMake can generate project files for Xcode and other popular IDEs.

Exiv2 has dependencies on the following libraries. All are optional, however it's unusual to build without zlib and libexpat.

Name	*Purpose*
zlib	Compression library. Necessary to support PNG files
expat	XML Library. Necessary to for XMP and samples/geotag.cpp
xmpsdk	Adobe Library for xmp. Source is embedded in the Exiv2 code base
libcurl	http, https, ftp, ftps support
libssh	ssh support
libiconv	charset transformations
libintl	localisation support

Conan

Starting with Exiv2 v0.27, we can use conan to build dependencies. I very much appreciate Luis bringing this technology to Exiv2 as it has hugely simplified building with Visual Studio. In the CI builders on GitHub, conan is also used to build on Linux and macOS. At this time (June 2020), I haven't been able to get conan to work on Cygwin and/or MinGW/msys2. I expect that will soon be rectified.

Prior to using conan, the build environment for Visual Studio was hand built for Visual Studio 2005 and relied on Visual Studio to convert to the edition in use. Additionally, source trees for dependencies were required in specific locations on storage. We did support CMake on Visual Studio, however it required a 500 line cmd file *cmakeBuild.cmd* to download and build the dependencies. The effort to create and maintain that apparatus was considerable.

Conan brings a fresh approach to building dependencies. Every dependancy has a "recipe" which tells conan how to build a dependency. The recipes are held on servers and are fetched from remote servers on demand. Exiv2 has a file conanfile.py (written in python) which tells conan which dependencies are required. Conan fetches the recipe, build the dependency and caches it for future use. When dealing with very large libraries such as openssl, the recipe might pull prebuilt binaries for your requested configuration. For more modest libraries (such as expat and zlib), the recipe will fetch the source and build. Conan caches binary dependencies in your ~/.conan directory. This arrangement means that you seldom build dependencies as they are usually in the cache.

I have always supported the plan to use CMake as our common build platform and invested a lot of effort in cmakeBuild.cmd. Using conan with Visual Studio is much superior to our prior methods of working with CMake and Visual Studio.

Luis has made other very useful contributions to Exiv2. He rewrote most of the CMake code. It's shorter, more robust, more flexible and easier to understand. He also introduced CPack which packages both our Source Bundle and binary builds for public release on https://exiv2.org

The final element of CMake which we have not yet deployed is CTest. Perhaps one day this will be investigated and integrated into the Exiv2 build environment.

Thank You to Luis for introducing Conan to Exiv2 and all your outstanding work with CMake.

The documentation about using Conan with Exiv2 is in README-CONAN.md

Build Options

There are numerous build options provided for Exiv2. The documentation for this is in README.md. Most of the options concern dependencies, the configuration:

Description	Choices
build_type	debug or release
kind	static or shared
configuration	32 or 64
C-runtime	shared or static
compiler	GCC or Clang or 2008 … 2019
language standard	98 or 11 or 14 or 17

And we have not considered the selection of build dependencies required by the user. For example, support for PNG, XMP, Localisation, Web Protocols and Character Conversions.

There are a number of convenience options to build packages for release, on-line documentation, unit_tests and ASAN support. ASAN is the "Address Sanitiser". When this option is selected, the code is built and instrumented with address checking. Every memory access is tested before use. This has a considerable performance penalty and is only intended for test and development purposes. It shouldn't be used in production.

An interesting option is BUILD_WITH_CCACHE. This option can dramatically speed up rebuilding by caching compiled code. If CCache determines that there are no code or configuration changes, the compiled object from the cache is used. This can accelerate build times by 100x.

While lots of effort has been invested in the CMakeLists.txt and *.cmake files, some users may want something that has never been investigated by Team Exiv2. For example, we do not support building for ARM processors. It's highly likely that Exiv2 can be successfully built for those machines and the recommended way is to use options such as -DCMAKE_CXX_FLAGS to introduce the necessary compiler and linker options. Other examples of "possible, yet not supported" are to request Visual Studio to use Clang, or its own CMake support, or its own Package Manager.

Regrettably there are users who look to Team Exiv2 to support every possible configuration. This is impossible. The essential thing is that we have built and tested our code on many platforms. Users will always think of novel ways in which to build and deploy. I worked with a user to build Exiv2 on OS/2. I had no idea that OS/2 is alive and well.

TOC

11.3 Security

This is discussed in detail here: 10 Security.

11.4 Documentation

The following types of documents used in Exiv2. They are:

	Documents	Creator	Comment
1	The exiv2 man page exiv2.1	man, man2html	Unix mark up syntax (troff)
2	User Manuals	Markdown	User Documentation
3	API Documentation	Doxygen	From .cpp and .hpp files
4	Web site pages	Scripted *(mostly)*	
5	Release Notes	Markdown	GitHub PRs
6	GitHub Wiki Pages	Markdown	https://github.com/Exiv2/exiv2.wiki.git

Life would be simpler with a single way to define documents and scripts to ***propagate*** changes to their destination. In some ways, this has been done. However the nature and format of the document classes are different, and the current arrangements will not yield to much more simplification.

Markdown

I'm very pleased by Markdown. Perhaps one day, the utility **man** will support this format as that would simplify the maintenance of exiv2.1.

This book has been written in markdown, and the User Documentation in the release. Markdown is also used extensively on GitHub for discussion and the project Wiki pages.

Doxygen generated API Documentation

The API documents are generated from comments in the C++ code. Doxygen generates UML diagrams of the class hierarchy, table of contents, navigation links and more. It does a very nice job with modest effort from Team Exiv2.

Release Documentation

Creating release notes takes quite a lot of time and effort. When Exiv2 v0.28 is released, the GitHub tools will probably do an adequate job. However while 0.27-maintenance and master are both developed, I feel manually creating the release notes is a better approach. The Release procedure is discussed here: 11.9 Releases

Exiv2 man page

I don't like the man page because it's in UNIX man page troff syntax which is arcane and unfamiliar. However man pages are very useful and valuable for users. The man page is stored in man/man1/exiv2.1 When editing the man pages, I inspect the changes with commands such as:

```Bash
$ cd \<exiv2dir\>
$ env MANPATH=$PWD/man:$MANPATH man exiv2 | grep EXIV2
EXIV2(1)                                                    EXIV2(1)
                                  Nov 6, 2020               EXIV2(1)
$
```

The man pages are converted with man2html for display on https://exiv2.org/manpage.html. In the release scripts, man page is converted to PostScript and then to PDF as follows:

```
$ env MANPATH=$EXIV2HOME/man:$MANPATH man -t p | ps2pdf
```

TOC

11.5 Testing.

This is discussed in detail here: 8 Test Suite.

11.6 Samples

Exiv2 has sample applications which have their own documentation: README-SAMPLES.md. In Exiv2 v0.27.3, there are 17 samples applications and 19 test programs. The test programs are intended for use by the test suite and are not installed on the user's computer.

The following programs are built and installed in /usr/local/bin.

Name	Purpose
addmoddel	Demonstrates Exiv2 library APIs to add, modify or delete metadata
exifcomment	Set Exif.Photo.UserComment in an image
exifdata	Prints ***Exif*** metadata in different formats in an image
exifprint	Print ***Exif*** metadata in images Miscelleous other features
exifvalue	Prints the value of a single ***Exif*** tag in a file
exiv2	Command line utility to read, write, delete and modify Exif, IPTC, XMP and ICC image metadata. This is the primary test tool used by Team Exiv2 and can exercise almost all code in the library. Due to the extensive capability of this utility, the APIs used are usually less obvious for casual code inspection.
exiv2json	Extracts data from image in JSON format. This program also contains a parser to recursively parse Xmp metadata into vectors and objects.
geotag	Reads GPX data and updates images with GPS Tags
iptceasy	Demonstrates read, set or modify IPTC metadata
iptcprint	Demonstrates Exiv2 library APIs to print Iptc data
metacopy	Demonstrates copying metadata from one image to another
mrwthumb	Sample program to extract a Minolta thumbnail from the makernote
taglist	Print a simple comma separated list of tags defined in Exiv2
xmpdump	Sample program to dump the XMP packet of an image
xmpparse	Read an XMP packet from a file, parse it and print all (known) properties.
xmpprint	Read an XMP from a file, parse it and print all (known) properties..
xmpsample	Demonstrates Exiv2 library high level XMP classes

Most of the programs are about 100 lines of C++ and do simple tasks to demonstrate how to use the library API. Three of the programs are substantial. They are: ***exiv2***, ***geotag*** and ***exiv2json***

The Exiv2 command-line program ***exiv2*** enables users to manipulate metadata in images using most of the features of the library. Being a general utility, it has about 4000 lines of code. The length of the program proves the point that it is full featured, however the quantity of code rather obscures the use of the library APIs.

Exiv2 has always resisted the temptation to provide a GUI version of the program as that would involve considerable cross-platform development and user interface skills. As Andreas Huggel summarised: *Exiv2 does*

depth, not breadth. Providing a GUI would lead the project away from metadata into the world of the *User Experience*.

TOC

11.7 Users

User Support is very time consuming. I prioritise working with users as the most important aspect of the project. Occasionally, in the run-up to a release, I will ask a user to wait. However, my default is to deal with users as quickly as possible. I try to acknowledge and confirm their report within 24 hours and to fix/close issues in one week.

The reason to give them priority is the importance of users to the project. Without users, the project is dead. Without support, users will not use the code. I know this is true because I have taken some sabbaticals to deal with other matters in my life. When I am not active, the number of user reports and requests falls quickly. When I return from my break, the number of user report immediately increases.

I have been very disappointed by the appreciation shown by users to my attention to their questions. Very few people have the courtesy to use words like "Please" and "Thank You". Why is this? I don't know. Moreover, I am astonished by the abuse I have encountered. The on-line behaviour of some users is unacceptable. I have encountered this behaviour from our OEM Engineers when I worked on Adobe PostScript. However, I could refer that to management at Adobe and at the OEM and the matter would settle. With open-source, I have to ignore and accept this awful behaviour.

I have wondered if the users who behave this way believe that I am a business and have let them down in some way. Open source is a community. In reporting a bug, they are participating in the development process. I usually thank users for reporting issues. It's sad that they seldom have the courtesy to thank me for fixing the issue.

A member of my family is the Principal of a College. We were discussing the behaviour of parents of students. She said *the one word you must never use with a parent is **No***. It's the same with open source users. It's pointless to say ***No*** because they will not accept this. A good example is Lens Recognition. The configuration file was added in 0.26 to enable users to fix lens recognition issues by updating an ascii file. Many users demand that I fix their lens in C++ to save them a minute to update ~/.exiv2. Saying ***No*** is pointless.

On a more positive note about dealing with users, I have enjoyed many on-line discussions with frequent visitors to exiv2.org. For sure, I include Arnold, Mikayel, Alan and Steve in this group and there are many more. If you are courteous, I am always pleased to hear from you. We are a community with a shared vision of working together. Thank You for participating.

TOC

11.8 Bugs

Exiv2 has used three bug tracking systems during its 17 year life. In the early days, issues were stored on a forum hosted by yahoo. (Who?). About the time that I joined the project (2008), Redmine was installed to track issues on exiv2.org. I really like Redmine. It has a nice UI with good search, cross referencing, and reporting tools. I very much appreciated the API to query and download data in JSON format. I had a script to generate

various report to monitor release progress.

We moved the code to GitHub when Exiv2 v0.26 was released in April 2017. I didn't know that GitHub provided issue tracking and many other project management tools. We could even consider closing exiv2.org in future and providing all project resources from GitHub.

I believe the GitHub Rest API provides a mechanism with which we could collect data. I could use that to generate report similar to my Remine/python/JSON code.

I'm pleased with GitHub. For sure, it's a "one stop shop" for a project. They provide good tools. The best aspect of GitHub is that I met Luis and Dan on GitHub. And numerous other frequent contributors (acknowledged on page 2 of this book). For sure GitHub has brought more order to the world of open-source.

TOC

11.9 Releases

Releases (both RCs and GMs) are published on GitHub. Users can receive notifications by subscribing an RSS reader to: https://github.com/exiv2/exiv2/releases.atom. There is a summary of releases here: https://github.com/Exiv2/exiv2/releases. All releases (both RCs and GMs) are available from exiv2.org at: https://exiv2.org/archive.html

Making a new release is very time-consuming. The business of performing the builds and updating the web-site is straightforward. It is totally scripted and easy to perform.

However the time involved in determining the contents of the release, updating the release notes, submitting all the PRs, testing and documenting is considerable.

Moreover, I like to publish release candidates. I never make code changes between the final release candidate the Golden Master. Let me define the terminology and the version numbering scheme.

Version	Name	Status	Purpose
0.27.7.3	Exiv2 v0.27.3	GM	Golden Master. This is the final and official release.
0.27.3.2	Exiv2 v0.27.3.2	RC2	Release Candidate 2.
v0.27.3.20	Exiv2 v0.27.3.2	RC2 Preview	Dry-run for release candidate. For team review.
v0.27.3.81	Exiv2 v0.27.3	Security Fix	Security Release
v0.27.3.29	Exiv2 v0.27.3.29	Development	Should never be installed for production.
v0.27.4.9	Exiv2 v0.27.4.9	Development	Should never be installed for production.
v0.27.99	Exiv2 v0.28	Development	Should never be installed for production.

The release procedure is documented here: svn://dev.exiv2.org/svn/team/website/Checklist.txt

It typically takes about 3 months to make a release and consumes 100-400 hours.

In month 1, the release and release notes are developed. Depending on the complexity of the features being

added for release, this can be 40 to 200 hours of work. In month 2, we respond to matters arising from RC1. As with month 1, it's usually 40-200 hours of work to reach RC2. In month 3, we do nothing. It's an afternoon's work to publish GM and tag the release.

If an issue arrives between RC2 and GM and it is decided to change code, I always accept a schedule delay and publish RC3.

It's only fair to say that others will say "Oh, it shouldn't be so complicated.". And I agree. It shouldn't. I've been the Release Engineer for at least 6 releases and have not discovered any tricks to eliminate the work involved. You could just tag the current development branch, bump the version number and hope for the best. While we currently have two major branches *0.27-maintenance* and *master*, this isn't possible. At least half the PRs and changes in the release are changes which have to be ported from the other branch.

If we reach Exiv2 v0.28, I hope that a further dot release from Exiv2 v0.27-maintenance will never be required. I suspect we will see Exiv2 v0.27.4 in 2021 and v0.27.5 in 2022 with security fixes which will need to be ported from *master*. To reach Exiv2 v0.28, there are numerous fixes in 0.27-maintenance which should be ported from 0.27-maintenance.

The Macro EXIV2_TEST_VERSION

This enables application code to easily test for a ***minimum*** version of exiv2.

```
#define EXIV2_TEST_VERSION(major,minor,patch) \
    ( EXIV2_VERSION >= EXIV2_MAKE_VERSION(major,minor,patch) )
```

For example, to safely call image->setIccProfile(), this is compile time safe and will not link Exiv2::Image::setIccProfile() if you are using any version of Exiv2 prior to 0.27.0 when this API was added.

```
#ifdef EXIV2_TEST_VERSION(0,27,0)
    image->setIccProfile(....);
#endif
```
C++

TOC

11.10 Platforms

There are several parts of Exiv2 which are platform specific. Additionally the platform dependent function *getopt()* in the C-runtime library is never used.

src/getopt.cpp

The command-line handler getopt() is used by the exiv2 command-line program and by samples/metacopy.cpp, samples/getopt-test.cpp and samples/toexv.cpp. In the early days of Exiv2, getopt() was provided by the C-runtime library. When support for msvc was added, the code in src/getopt.cpp was added. Relying on the C-runtime library revealed differences between platforms and between platforms and src/getopt.cpp. It was decided to ensure consistent behaviour to use src/getopt.cpp on all platforms.

src/version.cpp

I'm rather proud of the code in src/version.cpp. In addition to reporting the version number, version.cpp

reports the build settings used to compile the code. It also inspects the run-time environment to determine shared objects which have been loaded. I added this to the test harness because I was suspicious that we were not testing the correct shared object.

I'll write more later about how this is achieved.

src/futils.cpp

This file has utility code for dealing with files and paths. For example, there is a base64 encoder/decode which is used to manage paths of the form *data:abc…*. There is also a URL parser for decomposing URLs to determine protocol, serverer, user, password and other URL paraphernalia.

```Bash
.../exiv2/0.27-maintenance $ grep -e ') {' -e getProcessPath src/futils.cpp | grep
    char to_hex(char code) {
    char from_hex(char ch) {
    std::string urlencode(const char* str) {
    char* urldecode(const char* str) {
    void urldecode(std::string& str) {
    int base64encode(const void* data_buf, size_t dataLength, char* result, size_t resul
    long base64decode(const char *in, char *out, size_t out_size) {
    Protocol fileProtocol(const std::string& path) {
    Protocol fileProtocol(const std::wstring& path) {
    std::string pathOfFileUrl(const std::string& url) {
    std::wstring pathOfFileUrl(const std::wstring& wurl) {
    std::string getProcessPath()
.../exiv2/0.27-maintenance $
```

The function std::string getProcessPath() determines the process path and is similar to code in src/version.cpp.

UNICODE path support

There is a build option EXIV2_ENABLE_WIN_UNICODE which may be used on Windows when building with Visual Studio. This is useful for applications using wchar_t path strings. I believe this is the default for most applications using the Qt libraries. This version of the library can be used by other applications such as command-line utilities which link wmain(). When the library is build with UNICODE path support, the char versions of the API are also built.

Please be aware that this feature only applies to paths. Using UNICODE in UserComments and other Tags is explained in the Exiv2 man page and discussed in more detail below under the title ***Character Set Encoding***.

Here is a typical build sequence to build with UNICODE path support for Visual Studio:

```
>cd    <exiv2dir>
>mkdir build && cd build
build>conan install .. --profile msvc2019Release --build missing
build>cmake .. -G "Visual Studio 16 2019" -DEXIV2_ENABLE_WIN_UNICODE=On
build>cmake --build . --config Release
```

When the build finishes, you can inspect the setting and read the export table of ***ImageFactory::open()*** functions as follows. The output presented here has been simplified for presentation in this book.

```
>cd <exiv2dir>
>mkdir build && cd build
build>bin\exiv2 -vVg unicode
exiv2 0.27.3
processpath=C:\Users\rmills\gnu\github\exiv2\0.27-maintenance\build\bin
executable=C:\Users\rmills\gnu\github\exiv2\0.27-maintenance\build\bin\exiv2.exe
library=C:\Users\rmills\gnu\github\exiv2\0.27-maintenance\build\bin\exiv2.dll
have_unicode_path=1
build>dumpbin/exports bin\exiv2.dll | grep ImageFactory | grep open > foo.txt
Exiv2::Image::auto_ptr Exiv2::ImageFactory::open
... (class std::basic_string<char,struct std::char_traits<char>,class std::allocator<cha
...  const & __ptr64,bool)
Exiv2::Image::auto_ptr Exiv2::ImageFactory::open
...(class std::basic_string<wchar_t,struct std::char_traits<wchar_t>,class std::allocato
... const & __ptr64,bool)
Exiv2::Image::auto_ptr Exiv2::ImageFactory::open(unsigned char const * __ptr64,long)
Exiv2::Image::auto_ptr Exiv2::ImageFactory::open(class std::auto_ptr<class Exiv2::Basicl
```

Caution: *You should use the "Developer Command Prompt" in Visual Studio Studio to ensure you have the utilities dumpbin and undname on your path. To filter the output with grep, you will need to ensure grep is on your PATH.*

Performing the same tests on a default build (without UNICODE path support), shows one less entry point because the library does not provide the UNICODE entry-point ***Exiv2::ImageFactory::open(std::wstring,const bool&).***

The UNICODE library has been build with both wstring and string entry points. All samples (except exifprint.cpp) are built to use the char entry points, so you can the test suite runs.

```
build> cmake --build . --config Release --target tests
```

The test suite passes because our test image paths do not need UNICODE path support. See README.md for more information about running the test suite.

When you build the library with UNICODE path support, the sample program samples/exifprint.cpp is built as a UNICODE application. You can observe the UNICODE path support in the following way:

```
build>copy %USERPROFILE%\Stonehenge.jpg %USERPROFILE%\√.jpg
build>bin\exifprint.exe %USERPROFILE%\√.jpg | grep Date
Exif.Image.DateTime                          0x0132 Ascii      20  2015:02:18 21:42:57
Exif.Photo.DateTimeOriginal                  0x9003 Ascii      20  2015:02:15 12:29:49
Exif.Photo.DateTimeDigitized                 0x9004 Ascii      20  2015:02:15 12:29:49
Exif.NikonWt.DateDisplayFormat               0x0003 Byte        1  0
Exif.GPSInfo.GPSDateStamp                    0x001d Ascii      11  2015:02:15

build>cd ..\build_no_unicode
build_no_unicode>bin\exifprint.exe %USERPROFILE%\√.jpg | grep Date
build_no_unicode>
```

When you build the library with UNICODE path support, wchar_t versions of the following APIs are built:

```Bash
Exiv2::BasicError
Exiv2::FileIo::FileIo
Exiv2::HttpIo::HttpIo
Exiv2::XPathIo::XPathIo
Exiv2::ImageFactory::create
Exiv2::ImageFactory::createIo
Exiv2::fileExists
Exiv2::fileProtocol
Exiv2::ImageFactory::getType
Exiv2::ImageFactory::open
Exiv2::pathOfFileUrl
Exiv2::readFile
Exiv2::ExifThumb::setJpegThumbnail
Exiv2::FileIo::setPath
Exiv2::XPathIo::writeDataToFile
Exiv2::ExifThumbC::writeFile
Exiv2::writeFile
Exiv2::PreviewImage::writeFile
```

Character Set Encoding

This is discussed here: https://github.com/Exiv2/exiv2/issues/1258

Being a native English speaker, I find it difficult to understand the use of other character sets. I understand the importance of this to users, however I have no experience of typing anything other than 7-bit ascii.

The Exif specification for ASCII states:

2 = ASCII An 8-bit byte containing one 7-bit ASCII code. The final byte is terminated with NULL.

Exiv2 does not enforce the 7-bit condition. You can read/write any 8-bit value, including NUL. This is discussed: https://github.com/Exiv2/exiv2/issues/1279#issuecomment-689053734

You can use charset= on the 'Comment' tags which are:

```Bash
714 rmills@rmillsmbp:~/gnu/github/exiv2/0.27-maintenance/build $ taglist ALL | gre
Photo.UserComment,  37510,  0x9286, Photo,  Exif.Photo.UserComment, Comment
GPSInfo.GPSProcessingMethod,    27, 0x001b, GPSInfo,    Exif.GPSInfo.GPSProcessingMetho
GPSInfo.GPSAreaInformation, 28, 0x001c, GPSInfo,    Exif.GPSInfo.GPSAreaInformation,
715 rmills@rmillsmbp:~/gnu/github/exiv2/0.27-maintenance/build $
```

The format of Exif Comment values include an optional charset specification at the beginning. Comments are stored as Undefined tags with an 8-byte encoding definition follow by the encoded data. The charset is specified as follows:

```Bash
[charset=Ascii|Jis|Unicode|Undefined] comment
charset=Undefined is the default

$ exiv2 -M'set Exif.Photo.UserComment charset=Ascii My photo' x.jpg
$ exiv2 -pa --grep UserComment x.jpg
Exif.Photo.UserComment       Undefined  16  My photo
$ exiv2 -pv --grep UserComment x.jpg
0x9286 Photo     UserComment Undefined  16  charset=Ascii My photo

```

```
$ exiv2 -M'set Exif.Photo.UserComment charset=Unicode \u0052\u006f\u0062\u0069\u006e' x.
$ exiv2 -pa --grep UserComment x.jpg
Exif.Photo.UserComment             Undefined  18  Robin
$ exiv2 -pv --grep UserComment x.jpg
0x9286 Photo      UserComment             Undefined  18  charset=Unicode Robin

$ exiv2 -M'set Exif.GPSInfo.GPSProcessingMethod HYBRID-FIX' x.jpg
$ exiv2 -pa --grep ProcessingMethod  x.jpg
Exif.GPSInfo.GPSProcessingMethod           Undefined  18  HYBRID-FIX
$ exiv2 -pv --grep ProcessingMethod  x.jpg
0x001b GPSInfo   GPSProcessingMethod       Undefined  18  HYBRID-FIX
```

Exif Comments and characters outside the Basic Multilingual Plane

See: https://github.com/Exiv2/exiv2/issues/1279#issuecomment-689053734

IPTC and CharacterSet

IPTC Data Section 1 (Envelope) may include a Record 90 (CharacterSet). I know nothing about this record. It was briefly discussed here: https://github.com/Exiv2/exiv2/issues/1203

JSON Support

I really admire the code in samples/JZon.cpp. The Swedish Engineer who created this made a super-human effort to fix a bug for me. However, the latest public version of that code has lost his fix. I have spoken with him about this and we agreed that I would maintain the copy in the samples/. It isn't in the library. It is compiled and linked with samples/exiv2json.cpp

TOC

11.11 Localisation

Localisation is documented in README.md.

11.12 Build Servers

At different times, we have used different build server technologies.

1. Appveyor, Travis, GitLab and CodeCov
2. My build script ./build.sh
3. Jenkins and buildbot

Appveyor, Travis, GitLab and CodeCov

Those build systems are provided by GitHub and work very well. To use them, you have to check the appropriate box in the GitHub Branch Settings and add a configuration file to the branch. Exiv2 uses:

CI	Configuration	Comment
GitLib	.gitlab-ci.yml	Linux and UNIX
Travis	.travis.yml	Linux, macOS and UNIX
Appveyor	appveyor.yml	Visual Studio
Code Cov	codecov.yml	Linux Code Coverage

My build script ./build.sh

```Bash
503 rmills@rmillsmm-local:~/gnu/exiv2/team/contrib/buildserver $ svn info build.sh
URL: svn://dev.exiv2.org/svn/team/contrib/buildserver/build.sh
...
504 rmills@rmillsmm-local:~/gnu/exiv2/team/contrib/buildserver $ ./build.sh
usage: ./build.sh  { --help | -? | -h | platform  | switch | option     | location value

platform:    all[32] | msvc[32] | linux[32]  | macos      | cygwin | mingw | unix | free
switch:     --source | --debug  | --static   | --clang    | --background
options:   --[no]nls | --video  | --asan     | --status   | --[no]unit     | --[no]publi
msvc:         --2019 | --2017   | --2015     | --2013     | --2012         | --2010  | -
location: --server B | --user C | --builds D | --cpp  {98 | 11 | 14 | 17}  | --stamp st
           --github {rmillsmm,github,E}      | {--tag tag | --branch branch}
505 rmills@rmillsmm-local:~/gnu/exiv2/team/contrib/buildserver $
```

I abandoned Jenkins for several reasons:

1. It was insecure on the MacMini and could be hacked.
2. As configured by me, it was invoking a complicated ssh script.
3. I didn't understand how to trigger Jenkins from GitHub.
4. I didn't like the Cygwin ssh server which runs the bash shell.

I decided to forget about Jenkins and focus on the ssh script. So build.sh, parses its command arguments, writes the build script and transfers it by ssh to the appropriate VM. On the machine, rmillsmm, I have VM such as rmillsmm-w10, rmillsmm-ubuntu, rmillsmm-solaris and so on. On Windows, I use the excellant bitvise ssh server and the native server on other platforms.

Bitvise is a very solid server. https://www.bitvise.com/ssh-server. It can be configured for a variety of shells. I use cmd.exe to build on Visual Studio or Cygwin/64 or MinGW/msys2/mingw64. ./build.sh invokes the batch files cmd64.bat, cygwin64.bat or msys64.bat to configure the environment on the build machine. These scripts are discussed and documented in README.md.

Jenkins and buildbot

There are "out of the box" build servers. I evaluated both in 2013.

I didn't like Google's buildbot.

I used Jenkins for several years before deciding that it was not working well for me. Please understand that I have no criticism of Jenkins, my unhappiness was caused by my complicated Cygwin bash script. That script

has to invoke cmd.exe to build Visual Studio, then launch bash to run the test scripts.

I am glad to say that in Exiv2 v0.27.3, I studied the test environment and documented how to execute the bash scripts from cmd.exe on Windows. The 0.27-maintenance branch now has python code which makes it possible to remove bash from the test. So much of my unhappiness with Jenkins involved running the test suite and perhaps it's easier to configure Jenkins.

However, as GitHub provides good CI support, we don't need Jenkins.

TOC

11.13 Source Code

The source code for Exiv2 resides on GitHub https://github.com/exiv2/exiv2

There are team resources stored on subversion: svn://dev.exiv2.org/svn/team.

Here are most *(but not all)* of the team directories:

```Bash
731 rmills@rmillsmbp:~/gnu/exiv2/team $ ls -l
drwxr-xr-x+ 73 rmills  staff  2336 25 Oct 16:52 book              this book
drwxr-xr-x+  6 rmills  staff   192  4 Apr  2020 contrib           contributions (inclu
drwxr-xr-x+ 11 rmills  staff   352  8 Jun 12:02 drawings          miscellaneous artwor
drwxr-xr-x+  7 rmills  staff   224  3 Dec  2018 license           license discussions
drwxr-xr-x+ 24 rmills  staff   768 23 Jul 19:11 logo_files        artwork and fonts
drwxr-xr-x+  7 rmills  staff   224 21 Mar  2019 meetings          meeting minutes
drwxr-xr-x+ 98 rmills  staff  3136  8 Jul 11:26 releases          released source and b
drwxr-xr-x+  4 rmills  staff   128 19 May 12:03 rmills            my home-made scripts
drwxr-xr-x+ 15 rmills  staff   480  8 Jul 11:31 website           source for web site c
732 rmills@rmillsmbp:~/gnu/exiv2/team $
```

TOC

11.14 Web Site

The website source and release procedures are store in subversion. svn://dev.exiv2.org/svn/team. The release process is discussed in detail here: 11.9 Releases

TOC

11.15 Servers

We use several servers:

1. Apache (https://exiv2.org and https://pre-release.exiv2.org)
2. Subversion (svn://dev.exiv2.org/svn/)
3. GitHub (https://github.com/exiv2/exiv2)
4. Redmine (https://redmine.exiv2.org)
5. SSH on the buildserver

I am pleased to say that the management of exiv2.org is undertaken my Nehal. I don't think it's an onerous task, however I appreciate having this taken off my back.

TOC

11.16 API

This is discussed in detail here: 9. API/ABI Compatibility

11.17 Contributors

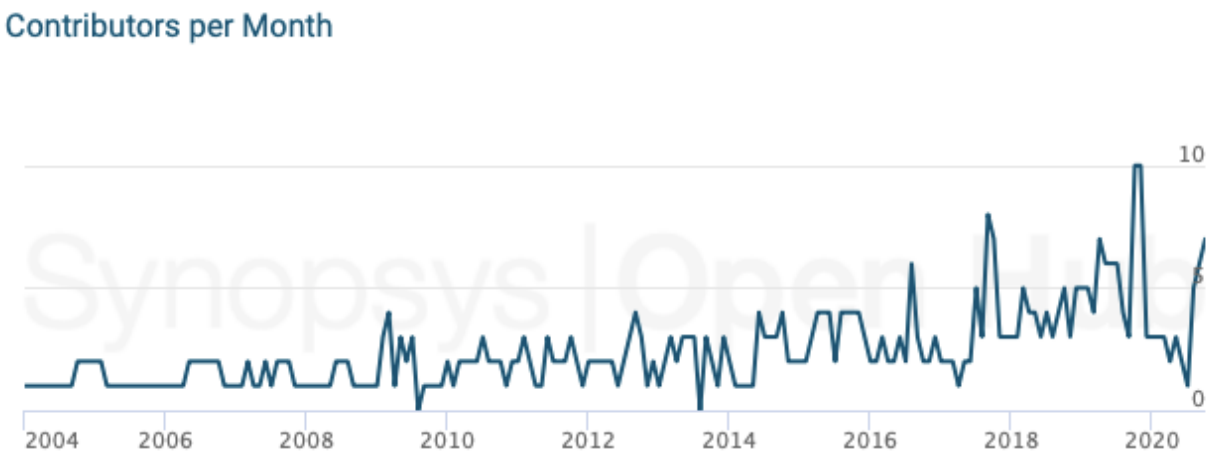
It's very difficult to recruit people to work on open source. In fact, it's so difficult that I wonder if open source can survive in future. Lots of people have made small contributions to Exiv2, however only a hand-full have made a sustained effort. Furthermore, contributors can disappear for months with no indication of their intention. I'm not criticising anybody for how they behave, however it's simply impossible to plan or schedule. When folks are paid in the office, you can reasonably expect that they will turn up regularly and can be assigned tasks. This model is invalid in open source.

I believe the large open source projects such as Apache, Clang and Mozilla employ engineers to undertake the work. I don't know how they are funded. However, when pay-checks are offered, recruitment is possible in the market.

A modest project such as Exiv2 has no money. In fact, I pay for the modest expenses such as hosting the web-site and running a build server on a Mac Mini.

The only major success I have had with recruitment is when Dan and Luis arrived in summer 2017. We adopted GitHub in April 2017 when Exiv2 v0.26 was released. I wondered if the move to GitHub had increased the project visibility and Dan and Luis would be the first of many contributors. More than 3 years later, we have enjoyed contributions from Kevin, Leo and Rosen.

I find the data on OpenHub very interesting: https://www.openhub.net/p/exiv2.

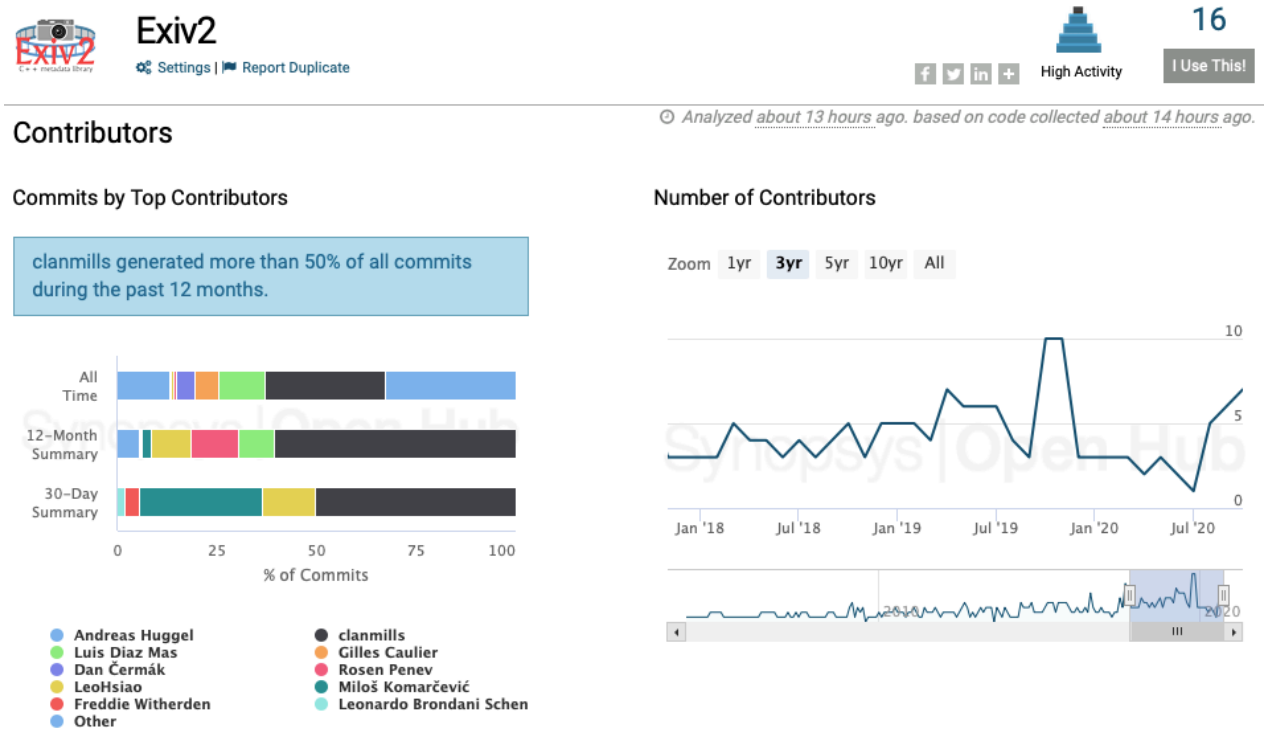


You can spot trends and events in the history:

Period	Comment	Period	Comment	Period	Comment
2014-2020	Growing	2013-14	I retired and moved home	2018-2019	Robin, Luis and Dan
2004-2008	Andreas	2016-2017	Robin and Ben	2019	I retired and contributors spiked
2009-2012	Andreas and Robin	2017	Moved to GitHub	2020	I returned for Exiv2 v0.27.3
2012-2015	Robin and Neils	2017	Dan and Luis Joined	2020	I wrote this book

The report provides interesting insight. Andreas remains the top contributor although I am catching him fast. We don't monitor the book or release script maintenance on OpenHub. My contributes to that put me well ahead of Andreas.

A boss once asked me "Do you know the **80/20** rule? **80**% of the project is done by **20**% of the people!". For sure, this is true in Open Source.



So, how are contributors recruited? The answer is ***I don't know***. For sure, I appreciate the work done by Andreas, Luis, Dan, Neils and about 20 other wonderful people. Curiously, I'm not aware of any lady contributors. I only recall one support question asked by a lady.

TOC

11.18 Scheduling

This is a major and important topic. Apart from writing code, I've spent more time thinking about project scheduling than any other aspect of Software Engineering.

There are two worlds. There is the perfect world which is inhabited by management. These people live in a world which is quite different from mine. In their world, the specification is clear, the schedule is realistic, nobody makes mistakes and everything works. It's a wonderful place. Sadly, I've never had the good fortune to live in that world.

I worked in a company which, to hide their identity, I will call "West Anchors". A colleague was giving a board presentation in which they had a slide:

It is the Policy of West Anchors to get it right first time, every time.

There we have it. Nothing is ever late, nothing is more difficult than expected, all suppliers deliver on time to specification and nobody is ever sick. When I discussed the project schedule with my boss I asked him why there was no time in the schedule for bugs and fixes, his response was "There better not be any bugs." Five years later, West Anchors were closed by their owners.

I also had the misfortune to work at a company where the boss was an expert in planning. He explained to me that the only challenge in Software Engineering is to get the schedule right. Everything else was trivial.

So, if you live in the perfect world, you'll not find anything interesting or useful in this part of the book, because I'm talking about the less than perfect world in which I live. I usually call it "Reality".

Another challenge is that many users are perfect and live in this other world where everything works. So users seldom understand that the open-source project may be populated by people who live in the depressing world of "Reality".

Scheduling an open-source project is almost impossible. You are dealing with volunteers. You might think you know the volunteers, however you don't. It's unusual to have even met the people. How can you understand the pressure and stress in another person's life when you know so little about their circumstances. And remember they are volunteers. They can walk off the job if they wish. In a business, management have tools such as reviews, salary, careers, vacations, bonuses, promotions and lay-offs to manipulate the employees. In the open-source world, you have none of those tools.

Here are my thought about how to solve the scheduling problem.

The Problem

The problem is really simple. How to plan large projects and deliver on time to budget.

The state of the game

Currently, planning is based mostly on PERT and descendant technology. Products such as Microsoft Project are designed to schedule resources and tasks. And indeed it works for some projects and fails hopelessly for others.

When the London 2012 Olympic Games were bid, the budget was $3 billion. The final cost has been stated as $20 billion. I have no data to say if there were other costs, such as policing, which are not included in the $20

billion.

This is rather common. The cost of construction of the aircraft carriers HMS Queen Elizabeth and HMS Prince of Wales are other high visibility projects in which the plan and reality are very different.

The reason for cost over-runs is that new work items are required that were not known early enough to be in the plan. We cannot know what we do not know. However there may be a way to calculate the size of the unknowns at the beginning of a project.

A project is recursive and requires recursive handling

When a project is simple - for example painting your house - it is possible to measure the size of the task and calculate the quantity of materials and labour required. This method works fine for a well defined project with quantifiable tasks.

However, if you want your house to be painted in an extra-ordinary way, this method totally fails. Think of Michael Angelo in the Sistine Chapel in Rome. Michael and the Pope came close to blows in a 20 year struggle to produce one of the wonders of man's creativity. Nobody gives a hoot today about the cost. Nobody cares about how long it took. Nobody can understand why the customer and the contractor were divided over something as trivial as money.

The reason for the cost over-runs is because the project is recursive. In a simple project, you have a sequence (or connected graph) of tasks:

	Task	**Task**	**Task**	
Begin	Remove furniture	Apply N litres of paint	Restore furniture	Done

If the project had many rooms (say 10-20), you have to schedule resources (people). You have a finite set of painters, and you may have more than 1 team of painters. However the basic linear model is not affected.

When you scale to painting something large (like an Aircraft Carrier), two items rapidly emerge to invalidate the simple model.

1) Requirements Change

The Aircraft Carrier requires stealth paint that hasn't been invented.

2) The paint task is large

You require training and inspection services to manage quality.

And many other things arrive which were not in the simple model. In the worst case, new tasks can be larger than the original task. You have an exploding, complex challenge.

To deal with this, you have to start a project inside the project. Something like "Remove furniture" is obvious in a house. But what would it mean on an Aircraft Carrier?

So, we stay calm and add more items to the project plan. And that's when and why everything goes wrong. The plan gets longer and more detailed. However it's still the same old linear model.

My observation is that the project is an assembly of projects. As you develop the project, every line item in the simple model is a project. And then there are projects inside the projects. For example if special paint is required for the Aircraft Carrier, that task is probably a complex network of projects involving chemistry, special machines to apply the paint, and maintenance processes for the ship in service.

What does this have to do with Fractals?

Everything. A project is a recursive entity that must be handled with recursive techniques. Fractal Geometry deals with recursion.

Being a retired Software Engineer, I have often been told "The project is late because you (the engineer) did not itemise the project properly at the outset.". Wrong. It's the inflexible PERT model that cannot handle recursion.

The State of Project Planning Today

The software industry has a huge and sad collection of projects which have come off the rails. If the bosses had known at the outset, things would have been different. There are two things we care about passionately:

1. How long is this going to take?
2. How much is this going to cost?

Notice, we don't get overly bothered about what "it" is. We care about time and money.

If we do not know about the special paint for the Aircraft Carrier, are we upset? No. However the cost and schedule damage is painful for everybody involved.

Now, we can't know what we do not know. Is there are a way to calculate the size of the unknowns? There might be, as I will explain.

When you plan a project, you say "How long did it take to do the last one?", take into account inflation and apply optimism "We won't make the same mistakes again.". This is very bad thinking. The United Kingdom has not built an Aircraft Carrier for almost 40 years. Most of the engineers working on HMS Queen Elizabeth were not born when HMS Invincible sailed to the Falklands.

A whole collection of project planning tools are now used in the software industry. Together they go under the banner: "Agile" or "Scrum". Scrum imposes a regime of meetings and reviews on the project team. Several of these techniques are interesting.

Measure	Description	Observation
Story Points Task Size Task Poker	The size of a task is not 1,2,3,4,5 as difficulty increases. They use the Fibonacci series: 1,2,3,5,8,13,21 … So big tasks rapidly increase their allocation of resources and time.	Fibonacci series is recursive: X(n) = X(n-1) + X(n-2) where X(1),X(2)=1
The Sprint Step wise linear	Scrum says “we can’t plan everything at once, however we can complete well defined tasks on sprint schedules (typically 2 weeks).	I don’t know how scrum deals with tasks that are longer than 1 sprint.
Velocity	The team velocity (average story points completed over the last 3 sprints) are monitored and used to verify that the team is being neither optimistic nor pessimistic in their determination of story points for tasks.	Velocity is not predicted, it is measured by project performance. In a nutshell, it is recursive.

However scrum has a fatal weakness. Nobody knows the size of the total project. The model is fundamentally inadequate, because it is a monitoring tool and not predictive.

Can we have a single unified model for project planning?

I believe there’s a measure in Fractals called "Roughness” which measures some feature of the recursive item. If you measure the roughness of animal lungs (which are of course recursive), they are about the same in Elephants, Humans and Mice. A value of 1.0 implies that the object is perfectly smooth. Higher values represent the chaotic nature of the item.

I think it’s possible to measure roughness in past projects in addition to the historical performance. So, although we have never built HMS Queen Elizabeth, we could know from other Naval projects:

1. How much paint/painter/per time unit (the only measure in Microsoft Project)
2. The roughness of painting Naval Ships (projects hiding inside the project)

Both are required to estimate the size of the task. PERT models assume a roughness of 1.0 and that is why it fails on large projects. No large project has a roughness of 1.0.

So how can we use this?

We need to do three things:

1. Add roughness to every item in the project plan
2. Collect data to estimate roughness
3. We need a pot of time and money, which I call ***Contingencies***

Contingencies are a % of the whole project that should be used to assign resources as sub-projects emerge. All items in the project should have contingencies from which additional resources can be allocated. This is non-confrontational and does not require blame and finger pointing. We knew about the roughness and must plan for it.

In the past, I have applied contingencies as big brush stokes to the complete project. If the project is similar to the last one, contingencies are 10%. If the project involves many unknowns, perhaps it is 300% of the project.

An ex-boss thought 414.159% Factor 3.14159 to walk round the object, then 1.0 for the task itself! The point is that when you do something for the first time, you will spend time doing work that is subsequently abandoned. Nothing new can be achieved without trial and error.

Research is required to measure task roughness in past projects to validate this approach.

Other serious limitations with PERT

There are serious limitations with PERT. I only intend to investigate the use of fractals in planning and to ignore other limitations of PERT such as:

Trouble	Observation
PERT assumes that you can itemise and quantify every task in the project.	If you are investigating something new, you can probably do neither of these things.
Many projects cannot be quantified.	Why isn't PERT used: 1. In crime investigation. 2. In medical treatment. 3. In investment management.
You will do abortive work and encounter road blocks.	Program Managers never plan time for this. Every innovative project has abortive work.
People are not interchangeable.	People leave, or are assigned to other projects. New team members require time to come up to speed with the project.
Some tasks have a gestation period.	If you are having a baby, more women won't reduce the 9 month wait. Adding people is often counter-productive.
Management, and other project stakeholders, change goals and objectives.	The circumstances surrounding the project can change and have major implications for the project.

Because of the recursive nature of projects, there are serious limitations hiding inside these limitations.

So what am I going to do about this?

When I retired, I was thinking about doing a PhD in this area and thought it might take 10,000 hours over 5 years. The only tasks that I could define in 2014 were:

1. Write an outline of the project
2. Find a University willing to mentor/supervise the effort
3. Learn all about Fractals
4. Research and publish a thesis
5. Graduate

What is the roughness of these tasks? Unknown. Graduate is simple. Or is it? Do I need a new kilt? Who's going to attend? Where will everybody stay? Even little tasks can grow into projects.

One thing is certain, getting a better approach to project estimation is of enormous importance. We have to do better. I have tried to set out here an area of investigation that is worthy of attention.

Final words about this. I didn't undertake a PhD. Instead I have spent 10,000 hours working on Exiv2. This book is my thesis. The presentation at LGM in Rennes is my defence. My reward is to know that I've done my best.

TOC

11.19 Enhancements

I'm not sure there is anything very interesting to be said about this. There are really different types of requests. For example, adding recognition for one lens may only require one line of C++, a test file and a 10-line python test script. This is straightforward and can be fixed within hours. At the other extreme is the request to support ISOBMFF files including HEIF and CR3. This project involves research, code, test, build and documentation changes. And to make it even more difficult, the Community have challenged the legality of providing the feature. This feature will take years to complete.

In principle, anybody can develop a feature and submit a PR. In reality, this seldom happens. When this does happen, the effort required by me and the developer is often about the same. So, being offered code in a PR often doubles my work-load.

TOC

11.20 Tools

Every year brings new/different tools. For example: cmake, git, MarkDown, conan and C++11. One of the remarkable properties of tools you have never used is that they are perfect and solve all known issues, *until you use them*. Tools you have never used are bug free and perfect. Or so I am told.

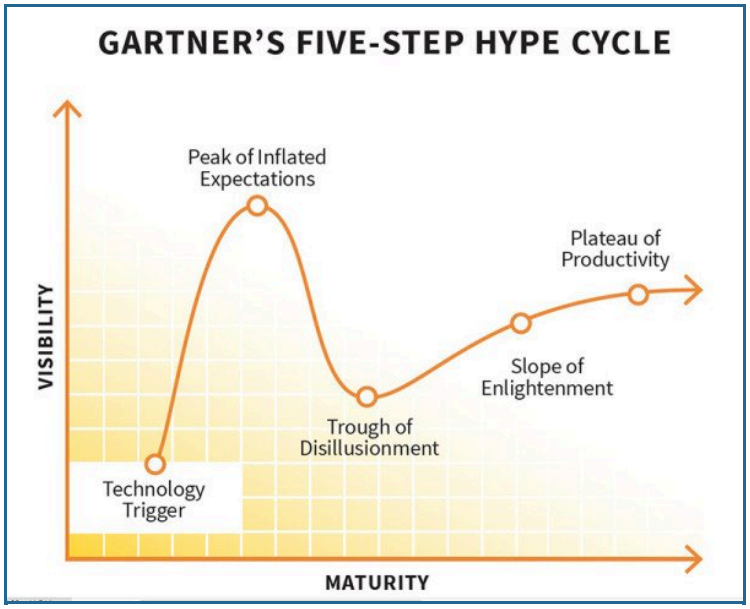
I had an issue with the release bundles for Exiv2 v0.26. My primary development platform is macOS. Remarkably, the version of tar shipped by Apple puts hidden files in bundles to store file extended attributes. I didn't know this until the bundles shipped and a bug report appeared. You cannot see those files on macOS, because tar on macOS recreates the extended attributes. However there were thousands of hidden files in the source bundle on Linux. I recreated the bundles as Exiv2 v0.27a and shipped them. There is an environment variable to suppress this. I believe it is: TAR_WRITER_OPTIONS=–no-mac-metadata.

Case closed. Except for very critical emails about changing bundles checksums.

For v0.27 we adopted CMake to do the packaging. Very nice. Works well. Guess what? CMake produces .tar.gz files which have these hidden files. Several people emailed to say "You wouldn't have this problem if you used CPack.". 100% wrong. It is a known documented issue in CPack. So, the issue resurfaced because we used CPack. Additionally, we had three release candidates for v0.27 which were published on 27 October, 15 November and 7 December 2018. v0.27 shipped on 20 December and the bug report arrived on the day after Christmas Day.

I rebuilt the bundles as Exiv2 v0.27.0a and shipped them on 2 January 2019. I updated the build script to ensure that source bundles are created on Linux.

Please understand that I have nothing against using new tools. However most of the hype surrounding new tools is nonsense. This has been studied. There are 5 stages in adopting new tools.



There is one recent tool which has surprised and pleased me. I have written this book using markdown and very pleased with the experience. As Americans say "your mileage may differ!".

TOC

11.21 Licensing

Licensing is a legal minefield. Exiv2 is licensed under GPLv2. Until Exiv2 v0.26, Andreas offered a commercial license for Exiv2. The contract between Andreas and users is not the concern of the Exiv2 open-source project.

In the days of the Commercial license, I made no distinction between open-source and commercial license users when it came to dealing with support and other requests. I felt that the commercial license freed the user from the obligations of GPL. However, it did not provide priority support, enhancement requests or any other benefit.

The general subject of the legality of Exiv2 hasn't been explored. There has been an enormous discussion about the legality of reading ISOBMFF files. See https://github.com/Exiv2/exiv2/issues/1229.

The ISOBMFF issue has caused me to wonder if Exiv2 is legal at all. I also wonder if any open source is legal! What makes something legal or illegal? Is everything legal until there is a law which declares it as illegal, or everything illegal until permitted by legislation? I suspect everything is legal until there is a legal precedent *legislation or court ruling* to the contrary.

Dealing with legal matters is not like reporting a bug. Exiv2 is an open-source project and we get a regular stream of issues reported on https://github.com/exiv2/exiv2. I acknowledge, investigate, reply and close the issue. By design, the process is focused on resolution. Legal processes are very different. When you ask for

legal advice, you are instigating an open-ended process which will endlessly expand.

TOC

11.22 Back-porting

I believe there are some folks maintaining back-ports of Exiv2. Our friend Pascal works on **darktable** and has back-ported many features and fixes. Thank You, Pascal for undertaking that chore.

I have to say that the inertia of the Linux Distros is considerable. It can take several years for new releases to arrive on the platform. I don't know anything about the distros and I'm not going to judge why it is so sluggish.

TOC

11.23 Partners

Without question, dealing with some other projects which use Exiv2 has been very difficult. Folks who have adopted Exiv2 in their product may feel they are entitled to make enhancement requests, demand fixes, superior support and other privileges. In a nutshell, they feel entitled. They are not. They are entitled to the same as all other stakeholders. No more. No less.

TOC

11.24 Development

As this is the first and last book I will ever write, I'd like to close the discussion of ***Project Management*** with some thoughts and opinions about how software is developed. Management have been searching for the silver bullet that will cause projects to deliver on time, to budget, with great performance, few bugs and low cost maintenance. This search has been proceeding for more than 50 years. We've made some progress. However system complexity out-strips our management and control tools. The challenges are immense.

I've seen different approaches used. In the IT world, people involved in system development adopted and modified the ***drawing office*** model. In the drawing office, you have draftsmen working on drawing boards and engineers working at desks. The engineers do the design and the draftsmen draw it. The systems analyst was the designer and the programmers created the code. They work in a strict regime of SSADM - the Standard Structure Analysis and Design Methodology. This is often called "The Waterfall Method". It's horrible. It's inflexible, slow and very expensive. It's amazing that anything can be delivered this way.

When I worked at West Anchors, the analysts promoted all the programmers to programmer/analyst. So the programmer had to do the programming and the work of the analyst. This enabled the analyst to concentrate on office politics. The QE team at West Anchors didn't test anything. They approved the test plans written by the programmer/analyst and they inspected the test logs required to prove that the programmer/analyst had done all the work. The parrot phrase of everybody who wasn't a programmer/analyst was "I'm not technical" which meant "I'm not going to help you, so don't ask. And, by the way, I am superior to you and you will do exactly what I tell you to do.".

Before I retired, the circus started adopting Scrum. Loads of meetings. The project is divided into two-week

sprints. There were two days of ***review*** meetings at the end of every sprint. Two days of ***planning*** meetings at the start of every sprint. Daily ***stand-up*** meetings which were usually about 1 hour. And I'm sure I've forgotten other pointless meetings. Sometimes people say they are ***agile***. I haven't figured out what that is. I think it's some kind of "Let's not bother looking ahead. It'll be great if or when it's delivered.". And of course, all software development engineers ***(except me)*** are geniuses who create perfect code and therefore have no reason to document or help lesser co-workers.

In the last 10 years we have seen AI move out of the lab and into our homes, cars and phones. Probably 50% of code development time is spent on test related activity. Perhaps in future, AI will undertake more of that work. Remember it works 7x24, never takes a vacation and works very quickly. I have high hopes that AI can be used to automate testing in future. However, all coins have two sides and the AI may drown the engineer with very obscure bugs. In some way, we see this with CVEs discovered by automatic fuzzing libraries.

There is a method of developing code that works for me and that's *to do everything myself*. This model doesn't scale. However it is effective. Do I create bugs? Of course, I do. However I find and fix them. Many of the best people with whom I worked in Silicon Valley use this approach. And when I think about it, that's exactly how Andreas created Exiv2.

Another method that I believe is very effective is prototyping. Working in a sand-box with a small amount of code can be very effective to explore and learn. I can say with certainty that I have learned more about metadata in 12 weeks by writing this book than I discovered by working on the Exiv2 code for 12 years. Program Management people hate prototyping because it doesn't have a specification, milestones, deliverables or schedule.

If you have good folks on the team, the development will be enjoyable and the results will be good. However, Software Development in large chaotic company such as *West Anchors* is Russian Roulette with a bullet in every chamber. Good Luck. I'm happy to be retired.

TOC

12 Code discussed in this book

The latest version of this book and the programs discussed are available for download from:

```Bash
svn://dev.exiv2.org/svn/team/book
```

To download and build these programs:

```Bash
$ svn export svn://dev.exiv2.org/svn/team/book
$ mkdir book/build
$ cd book/build
$ cmake ..
$ make
```

I strongly encourage you to download, build and install Exiv2. The current (and all earlier releases) are available from: https://exiv2.org.

There is substantial documentation provided with the Exiv2 project. This book does not duplicate the project documentation, but compliments it by explaining how and why the code works.

The following two programs args.cpp and dmpf.cpp are based on similar utility programs on the Apollo Workstatation on which I worked during the 1980s.

make test

The code in the book has a simple test harness in test/run.sh. When you build, you can run the tests with the command:

```Bash
586 rmills@rmillsmbp:~/gnu/exiv2/team/book/build $ make tests
Scanning dependencies of target tests
20200717_221452.avif passed
args passed
avi.avi passed
avif.avif passed
Canon.cr2 passed
Canon.crw passed
Canon.jpg passed
cr3.cr3 passed
csv passed
dmpf passed
heic.heic passed
IMG1.HEIC passed
IMG_3578.HEIC passed
mrw.mrw passed
NEF.NEF passed
NikonD5300.dcp passed
ORF.ORF passed
Stonehenge.jpg passed
Stonehenge.tiff passed
webp.webp passed
--------------------
Passed 20 Failed 0
--------------------
Built target tests
587 rmills@rmillsmbp:~/gnu/exiv2/team/book/build $
```

The code to implement the tests is in test/run.sh

```bash
#!/usr/bin/env bash

pass=0
fail=0

# Create reference and tmp directories
if [ ! -e ../test/data ]; then mkdir ../test/data ; fi
if [ ! -e ../test/tmp  ]; then mkdir ../test/tmp  ; fi

report()
{
    stub=$1
    # if there's no reference file, create one
    # (make it easy to add tests or delete and rewrite all reference files)
    if [ ! -e "../test/data/$stub" ]; then
        cp "../test/tmp/$stub" ../test/data
    fi

    diff -q "../test/tmp/$stub" "../test/data/$stub" >/dev/null
    if [ "$?" == "0" ]; then
        echo "$stub passed";
        pass=$((pass+1))
    else
        echo "$stub failed"
        fail=$((fail+1))
    fi
}

# test every file in ../files
for i in $( ls ../files/* | sort --ignore-case ) ; do
    stub=$(basename $i)
    # dmpf and csv are utility tests
    if [ $stub == dmpf -o $stub == csv -o $stub == args ]; then
        ./$stub ../files/$stub 2>&1 > "../test/tmp/$stub"
    else
        ./tvisitor -pRU "$i"   2&>1 > "../test/tmp/$stub"
    fi
    report $stub
done

echo --------------------
echo Passed $pass Failed $fail
echo --------------------

# That's all Folks
##
```

The CMake code in CMakeLists.txt is:

```
# Test harness (in ../test)
add_custom_target(test  COMMAND ../test/run.sh )
add_custom_target(tests COMMAND ../test/run.sh )
```

args.cpp

The purpose of this program is to analyse command-line arguments.

```C++
#include <stdio.h>
int main(int argc, char* argv[])
{
    int i = 1 ;
    while ( i < argc ) {
        printf("%-2d: %s\n",i,argv[i]) ;
        i++;
    }
    return 0 ;
}
```

csv.cpp

The purpose of this program is to "pretty print" csv files.

```C++
// http://www.zedwood.com/article/cpp-csv-parser
#include <string>
#include <vector>
#include <iostream>
#include <fstream>
#include <sstream>
#include <istream>

std::vector<std::string> read(std::istream &in, char delimiter)
{
    std::stringstream ss;
    bool inquotes = false;
    bool bEnd     = false;
    char Q = '"'  ; // quote character
    char L = '\n' ; // line-feed
    char C = '\r' ; // carriage-return

    std::vector<std::string> row;

    while(in.good() && !bEnd) {
        char c = in.get();
        if (!inquotes && c==Q) {
            inquotes=true;
        } else if (inquotes && c==Q) {
            if ( in.peek() == Q) { //2 consecutive quotes resolve to 1
                ss << (char)in.get();
            } else { //endquotechar
                inquotes=false;
            }
        } else if (!inquotes && c==delimiter) { //end of field
            row.push_back( ss.str() );
            ss.str("");
        } else if (!inquotes && (c==C || c==L) ) {
            if(in.peek()==L) { in.get(); }
            row.push_back( ss.str() );
            bEnd = true;
        } else {
          ss << c;
        }
```

```
    }
    return row;
}

int main(int argc, char *argv[])
{
    if ( argc != 2 ) {
        std::cerr << "usage: " << argv[0] << " { path | - }" << std::endl;
        return 1;
    }

    // open file and connect to std::cin
    std::string   path(argv[1]);
    std::ifstream file(path);
    if ( path != "-" ) {
        if ( file.is_open() ) {
            std::cin.rdbuf(file.rdbuf());
        } else if ( argc > 1 ) {
            std::cerr << "file did not open: " << path << std::endl;
            return 2;
        }
    }

    // parse input line by line
    while( std::cin.good() )
    {
        std::vector<std::string> row = read(std::cin , ',');
        for(int i=0, leng=row.size(); i<leng; i++)
            std::cout << "[" << row[i] << "]" << "\t";
        std::cout << std::endl;
    }

    file.close();
    return 0;
}
```

dmpf.cpp

The purpose of this program is to inspect files. It's ***od*** on steroids.

```C++
// g++ --std=c++11 dmpf.cpp
#include <stdio.h>
#include <map>
#include <string.h>
#include <vector>
#include <string>
#include <cstring>
#include <iostream>
#include <sstream>

#ifdef _MSC_VER
#pragma warning(disable : 4996)
#endif

std::vector      <std::string> paths;
std::string                    terminal("-");
```

```
std::map<std::string,uint32_t> options;

static enum error_e
{    errorOK = 0
,     errorSyntax
,   errorProcessing
}   error = errorOK ;

uint8_t print(uint8_t c) { return c >= 32 && c < 127 ? c : c==0 ? '_' : '.' ; }

void printOptions(error_e e)
{
    if ( options["verbose"] || e == errorSyntax ) {
        size_t count=0;
        for ( auto option : options ) {
            std::cout << (count++?" ":"") << option.first << "=" << option.second << ""
        }
    }
    std::cout << std::endl;
    error = e;
}

void syntax(int argc, char* argv[],error_e e)
{
    std::cout << "syntax: " << argv[0] << " [key=value]+ path+" << std::endl;
    printOptions(errorSyntax) ;
}

bool split(const char* arg,std::string& key,uint32_t& value)
{
    while ( *arg == '-' ) arg++;
    const char* chop = std::strchr(arg,'=');
    if ( chop ) {
        key   = std::string(arg,chop-arg);
        value = atoi(chop+1);
    }
    return chop != NULL;
}

// endian and byte swappers
bool isPlatformBigEndian()
{
    union { uint32_t i; char c[4]; } e = { 0x01000000 };
    return e.c[0]?true:false;
}

uint32_t platformEndian() { return isPlatformBigEndian() ? 1 : 0; }
void swap(void* from,void* to,size_t n)
{
    uint8_t*     v = reinterpret_cast<uint8_t *>(from);
    uint8_t*  swap = reinterpret_cast<uint8_t *>(to  );
    for (size_t i = 0; i < n; i++) {
        swap[i] = v[n - i - 1];
    }
}
uint64_t swap(uint64_t* value,bool bSwap)
```

```
{
    uint64_t result = *value ;
    if ( bSwap ) swap(value,&result,sizeof result);
    return result;
}
uint32_t swap(uint32_t* value,bool bSwap)
{
    uint32_t result = *value ;
    if ( bSwap ) swap(value,&result,sizeof result);
    return result;
}
uint16_t swap(uint16_t* value,bool bSwap)
{
    uint16_t result = *value ;
    if ( bSwap ) swap(value,&result,sizeof result);
    return result;
}

std::vector<std::string> splitter (const std::string &s, char delim)
{
    std::vector<std::string> result;
    std::stringstream        ss (s);
    std::string              item;

    while (getline (ss, item, delim)) {
        result.push_back (item);
    }

    return result;
}

bool file(const char* arg,std::string& stub,uint32_t& skip)
{
    std::string path(arg);
    if ( path == terminal ) { stub = terminal ; return true ; }

    // parse path/to/file[:number+length]+
    std::vector<std::string> paths = ::splitter(path,':');
    for ( size_t i = 1 ; i < paths.size() ; i++ ) {
        std::vector<std::string> numbers = splitter(paths[i],'+');
        skip += ::atoi(numbers[0].c_str());
    }
    FILE*  f      = ::fopen(paths[0].c_str(),"rb");
    bool   result = f != NULL ;
    if     (f) fclose(f);
    stub=paths[0];
    return result;
} //file

int main(int argc, char* argv[])
{
    options["bs"     ] =  1;
    options["width"  ] = 32;
    options["count"  ] =  0;
    options["endian" ] =  isPlatformBigEndian();
    options["hex"    ] =  1;
```

```
129     options["skip"   ] =  0;
130     options["verbose"] =  0;
131     options["start"  ] =  0; // set by file[:start->length]+
132
133     // parse arguments
134     if ( argc < 2 ) {
135         syntax(argc,argv,errorSyntax) ;
136     } else for ( int i = 1 ; i < argc ; i++ ) {
137         const char* arg = argv[i];
138         std::string key;
139         std::string stub;
140         uint32_t    value ;
141         bool        bClaimed = false;
142         if ( split(argv[i],key,value) ) {
143             if ( options.find(key) != options.end() ) {
144                 options[key]+=value;
145                 bClaimed = true ;
146             }
147         } else if ( file(arg,stub,options["start"]) ) {
148             paths.push_back(stub);
149             bClaimed = true;
150         }
151         if ( !bClaimed ) {
152             std::cerr << "argument not understood: " << arg << std::endl;
153             error = errorProcessing;
154         }
155     }
156
157     // report arguments
158     if ( options["verbose"] ) printOptions(error) ;
159
160     // process
161     if ( !error  ) for ( auto path : paths ) {
162         FILE* f = NULL  ;
163         size_t  size   = 1;
164         size_t  skip   = options["skip" ];
165         size_t  count  = options["count"];
166         size_t  width  = options["width"];
167         size_t  start  = options["start"];
168
169         std::cout << "path = " << path << std::endl;
170
171         if ( path != terminal ) {
172             f     = fopen(path.c_str(),"rb");
173             fseek(f,0,SEEK_END);
174             size  = ftell(f);
175         } else {
176             f      = stdin   ;
177             size   = 256*1024;
178         }
179         if ( !count ) count = size - skip-start;
180         if ( !f || (skip+count+start) > size ) {
181             std::cerr << path << " insufficient data" << std::endl;
182             error = errorProcessing;
183         }
184
185         char    line[1000]  ;
```

```
186         char    buff[64]    ;
187         size_t  reads  = 0 ; // count the reads
188         size_t  nRead  = 0 ; // bytes actually read
189         size_t  remain = count ; // how many bytes still to read
190         if ( width > sizeof buff ) width = sizeof(buff);
191         fseek(f,(long)skip+start,SEEK_SET);
192
193         if ( !error ) while ( remain && (nRead = fread(buff,1,remain>width?width:remain,
194             // line number
195             int l = sprintf(line,"%#8lx %8ld: ",(unsigned long)(skip+reads*width), (unsi
196
197             // ascii print
198             for ( int i = 0 ; i < nRead ; i++ ) {
199                 l += sprintf(line+l,"%c", print(buff[i])) ;
200             }
201
202             // blank pad the ascii
203             size_t  n   = nRead ;
204             while ( n++ <  width ) {
205                 l += sprintf(line+l," ") ;
206             }
207             l += sprintf(line+l,"  -> ") ;
208
209             size_t   bs = options["bs"];
210             switch ( bs ) {
211             case 8 :
212                 for ( size_t i = 0 ; i < nRead; i += bs ) {
213                     uint64_t* p = (uint64_t*) &buff[i] ;
214                     uint64_t  v = swap(p, options["endian"]!=platformEndian() );
215                     l += options["hex"] ? sprintf(line+l," %16llx" ,(long long int)v )
216                                         : sprintf(line+l," %20lld" ,(long long int)v )
217                     ;
218                 }
219             break;
220             case 4 :
221                 for ( size_t i = 0 ; i < nRead ; i += bs ) {
222                     uint32_t* p = (uint32_t*) &buff[i] ;
223                     uint32_t  v = swap(p,  options["endian"]!=platformEndian());
224                     l += options["hex"] ? sprintf(line+l,"  %8x" ,v )
225                                         : sprintf(line+l," %10d" ,v )
226                     ;
227                 }
228             break;
229             case 2:
230                 for ( size_t i = 0 ; i < nRead ; i += bs ) {
231                     uint16_t* p = (uint16_t*) &buff[i] ;
232                     uint16_t  v = swap(p,  options["endian"]!=platformEndian());
233                     l += options["hex"] ? sprintf(line+l," %4x" ,v )
234                                         : sprintf(line+l," %5d" ,v )
235                     ;
236                 }
237             break;
238             default:
239                 for ( int i = 0 ; i < nRead ; i++ ) { // bs == 1
240                     uint8_t v = buff[i];
241                     l += options["hex"] ? sprintf(line+l," %02x" ,v )
```

```
242                                     : sprintf(line+l," %3d" ,v )
243                     ;
244                 }
245             }
246
247             line[l] = 0 ;
248             std::cout << line << std::endl;
249             reads++;
250             remain -= nRead;
251             if ( path == terminal ) size += nRead;
252         } // while remains && nRead
253
254         if ( f != stdin ) {
255             fclose(f);
256         }
257         f = NULL;
258     }
259
260     return error ;
261 } // main
```

TOC

The Last Word

I hope you found this book interesting. More to the point, I hope you found the book useful. I hope Exiv2 will live into the future, or this book inspires somebody to write a new library.

I'm going off to cut the grass and to run in the beautiful countryside around my home in Camberley, England. And I'm going to play the Euphonium and the Piano. If you have interesting and positive thoughts you are welcome to open an issue on GitHub and I will respond. https://github.com/exiv2/exiv2

TOC