glymur Documentation

Release 0.8.18

John Evans

Nov 16, 2019

Contents

1 Glymur: a Python interface for JPEG 2000 **3**
- 1.1 Glymur Installation . . . 3

2 Advanced Installation Instructions **5**
- 2.1 Glymur Configuration . . . 5

3 How do I. . . ? **7**
- 3.1 . . . read images? . . . 7
- 3.2 . . . make use of OpenJPEG's thread support to read images? . . . 7
- 3.3 . . . write images? . . . 8
- 3.4 . . . write images with different compression ratios for different layers? . . . 8
- 3.5 . . . write images with different PSNR (or "quality") for different layers? . . . 8
- 3.6 . . . display metadata? . . . 9
- 3.7 . . . add XML metadata? . . . 12
- 3.8 . . . add metadata in a more general fashion? . . . 13
- 3.9 . . . work with ICC profiles? . . . 14
- 3.10 . . . create an image with an alpha layer? . . . 14
- 3.11 . . . work with XMP UUIDs? . . . 17

4 "What's new" documents **21**
- 4.1 Changes coming in 0.9 . . . 21
- 4.2 Changes in glymur 0.8 . . . 21
- 4.3 Changes in glymur 0.7 . . . 24
- 4.4 Changes in glymur 0.6 . . . 24
- 4.5 Changes in glymur 0.5 . . . 25

5 Known Issues **27**

6 Roadmap **29**

7 Indices and tables **31**

Contents:

CHAPTER 1

Glymur: a Python interface for JPEG 2000

Glymur is an interface to the OpenJPEG library which allows one to read and write JPEG 2000 files from Python. Glymur supports both reading and writing of JPEG 2000 images, but writing JPEG 2000 images is currently limited to images that can fit in memory. **Glymur** can read images using OpenJPEG library versions as far back as 1.3, but it is strongly recommended to use at least version 2.1.2.

In regards to metadata, most JP2 boxes are properly interpreted. Certain optional JP2 boxes can also be written, including XML boxes and XMP UUIDs. There is incomplete support for reading JPX metadata.

Glymur will look to use **lxml** when processing boxes with XML content, but can fall back upon the standard library's **ElementTree** if **lxml** is not available.

Glymur works on Python versions 2.7, 3.5, 3.6, 3.7, and 3.8.

For more information about OpenJPEG, please consult http://www.openjpeg.org.

1.1 Glymur Installation

The easiest way to install Glymur is via Anaconda using conda-forge

```
$ conda create -n testglymur -c conda-forge python glymur
$ conda activate testglymur
```

CHAPTER 2

Advanced Installation Instructions

2.1 Glymur Configuration

If you installed OpenJPEG via conda, you don't have to do any configuration, as glymur can find the OpenJPEG library within the Anaconda directory structure.

Otherwise, the default glymur installation process relies upon OpenJPEG being properly installed on your system as a shared library. If you have OpenJPEG installed through your system's package manager on linux, Cygwin, or if you use MacPorts on the mac, you are probably already set to go. But if you have OpenJPEG installed into a non-standard place or if you use windows, then read on.

Glymur uses ctypes to access the openjp2/openjpeg libraries, and because ctypes accesses libraries in a platform-dependent manner, it is recommended that **if** you compile and install OpenJPEG into a non-standard location, you should then create a configuration file to help Glymur properly find the openjpeg or openjp2 libraries The configuration format is the same as used by Python's configparser module, i.e.

```
[library]
openjp2: /somewhere/lib/libopenjp2.so
```

This assumes, of course, that you've installed OpenJPEG into /somewhere/lib on a linux system. The location of the configuration file can vary as well. If you use either linux or mac, the path to the configuration file would normally be

```
$HOME/.config/glymur/glymurrc
```

but if you have the **XDG_CONFIG_HOME** environment variable defined, the path will be

```
$XDG_CONFIG_HOME/glymur/glymurrc
```

On windows, the path to the configuration file can be determined by starting up Python and typing

```
import os
os.path.join(os.path.expanduser('~'), 'glymur', 'glymurrc')
```

You may also include a line for the version 1.x openjpeg library if you have it installed in a non-standard place, i.e.

```
[library]
openjpeg: /somewhere/lib/libopenjpeg.so
```

Once again, you should not have to bother with a configuration file if you use mac, linux, or Cygwin, and OpenJPEG is provided by your package manager.

CHAPTER 3

How do I. . . ?

3.1 . . . read images?

Jp2k implements slicing via the `__getitem__()` method and hooks it into the multiple resolution property of JPEG 2000 imagery. This means that lower-resolution imagery can be accessed via array-style slicing that utilizes strides. For example here's how to retrieve a full resolution and first lower-resolution image

```
>>> import glymur
>>> jp2file = glymur.data.nemo() # just a path to a JPEG2000 file
>>> jp2 = glymur.Jp2k(jp2file)
>>> fullres = jp2[:]
>>> fullres.shape
(1456, 2592, 3)
>>> thumbnail = jp2[::2, ::2]
>>> thumbnail.shape
(728, 1296, 3)
```

3.2 . . . make use of OpenJPEG's thread support to read images?

If you have glymur 0.8.13 or higher and OpenJPEG 2.2.0 or higher, you can make use of OpenJPEG's thread support to speed-up read operations

```
>>> import glymur
>>> import time
>>> jp2file = glymur.data.nemo()
>>> jp2 = glymur.Jp2k(jp2file)
>>> t0 = time.time(); data = jp2[:]; t1 = time.time()
>>> t1 - t0
0.9024193286895752
>>> glymur.set_option('lib.num_threads', 2)
>>> t0 = time.time(); data = jp2[:]; t1 = time.time()
```

(continues on next page)

(continued from previous page)

```
>>> t1 - t0
0.4060473537445068
```

3.3 . . . write images?

It's pretty simple, just supply the image data as a keyword argument to the Jp2k constructor.

```
>>> import glymur, numpy as np
>>> jp2 = glymur.Jp2k('zeros.jp2', data=np.zeros((640, 480), dtype=np.uint8))
```

You must have OpenJPEG version 1.5 or more recent in order to write JPEG 2000 images with glymur.

3.4 . . . write images with different compression ratios for different layers?

Different compression factors may be specified with the cratios parameter

```
>>> import skimage.data, glymur
>>> data = skimage.data.camera()
>>> # quality layer 1: compress 20x
>>> # quality layer 2: compress 10x
>>> # quality layer 3: compress lossless
>>> jp2 = glymur.Jp2k('myfile.jp2', data=data, cratios=[20, 10, 1])
>>> # read the lossless layer
>>> jp2.layer = 2
>>> data = jp2[:]
```

3.5 . . . write images with different PSNR (or "quality") for different layers?

Different PSNR values may be specified with the psnr parameter. Please read https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio for a basic understanding of PSNR.

Values must be increasing, but the last value may be 0 to indicate the layer is lossless. However, the OpenJPEG library will reorder the layers to make the first layer lossless, not the last.

```
>>> import skimage.data, skimage.measure, glymur
>>> truth = skimage.data.camera()
>>> jp2 = glymur.Jp2k('myfile.jp2', data=truth, psnr=[30, 40, 50, 0])
>>> psnr = []
>>> for layer in range(4):
...     jp2.layer = layer
...     psnr.append(skimage.measure.compare_psnr(truth, jp2[:]))
>>> print(psnr)
[inf, 29.028560403833303, 39.206919416670402, 47.593129828702246]
```

3.6 ... display metadata?

There are two ways. From the command line, the console script **jp2dump** is available.

```
$ jp2dump /path/to/glymur/installation/data/nemo.jp2
```

From within Python, the same result is obtained simply by printing the Jp2k object, i.e.

```
>>> import glymur
>>> jp2file = glymur.data.nemo() # just a path to a JP2 file
>>> jp2 = glymur.Jp2k(jp2file)
>>> print(jp2)
File:  nemo.jp2
JPEG 2000 Signature Box (jP  ) @ (0, 12)
    Signature:  0d0a870a
File Type Box (ftyp) @ (12, 20)
    Brand:  jp2
    Compatibility:  ['jp2 ']
JP2 Header Box (jp2h) @ (32, 45)
    Image Header Box (ihdr) @ (40, 22)
        Size:  [1456 2592 3]
        Bitdepth:  8
        Signed:  False
        Compression:  wavelet
        Colorspace Unknown:  False
    Colour Specification Box (colr) @ (62, 15)
        Method:  enumerated colorspace
        Precedence:  0
        Colorspace:  sRGB
UUID Box (uuid) @ (77, 3146)
    UUID:  be7acfcb-97a9-42e8-9c71-999491e3afac (XMP)
    UUID Data:
    <ns0:xmpmeta xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:ns0="adobe:ns:meta/
↪" xmlns:ns2="http://ns.adobe.com/xap/1.0/" xmlns:ns3="http://ns.adobe.com/tiff/1.0/
↪" xmlns:ns4="http://ns.adobe.com/exif/1.0/" xmlns:ns5="http://ns.adobe.com/
↪photoshop/1.0/" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" ns0:xmptk=
↪"Exempi + XMP Core 5.1.2">
      <rdf:RDF>
        <rdf:Description rdf:about="">
          <ns2:CreatorTool>Google</ns2:CreatorTool>
          <ns2:CreateDate>2013-02-09T14:47:53</ns2:CreateDate>
        </rdf:Description>
        <rdf:Description rdf:about="">
          <ns3:YCbCrPositioning>1</ns3:YCbCrPositioning>
          <ns3:XResolution>72/1</ns3:XResolution>
          <ns3:YResolution>72/1</ns3:YResolution>
          <ns3:ResolutionUnit>2</ns3:ResolutionUnit>
          <ns3:Make>HTC</ns3:Make>
          <ns3:Model>HTC Glacier</ns3:Model>
          <ns3:ImageWidth>2592</ns3:ImageWidth>
          <ns3:ImageLength>1456</ns3:ImageLength>
          <ns3:BitsPerSample>
            <rdf:Seq>
              <rdf:li>8</rdf:li>
              <rdf:li>8</rdf:li>
              <rdf:li>8</rdf:li>
            </rdf:Seq>
```

(continues on next page)

(continued from previous page)

```
          </ns3:BitsPerSample>
          <ns3:PhotometricInterpretation>2</ns3:PhotometricInterpretation>
          <ns3:SamplesPerPixel>3</ns3:SamplesPerPixel>
          <ns3:WhitePoint>
            <rdf:Seq>
              <rdf:li>1343036288/4294967295</rdf:li>
              <rdf:li>1413044224/4294967295</rdf:li>
            </rdf:Seq>
          </ns3:WhitePoint>
          <ns3:PrimaryChromaticities>
            <rdf:Seq>
              <rdf:li>2748779008/4294967295</rdf:li>
              <rdf:li>1417339264/4294967295</rdf:li>
              <rdf:li>1288490240/4294967295</rdf:li>
              <rdf:li>2576980480/4294967295</rdf:li>
              <rdf:li>644245120/4294967295</rdf:li>
              <rdf:li>257698032/4294967295</rdf:li>
            </rdf:Seq>
          </ns3:PrimaryChromaticities>
        </rdf:Description>
        <rdf:Description rdf:about="">
          <ns4:ColorSpace>1</ns4:ColorSpace>
          <ns4:PixelXDimension>2528</ns4:PixelXDimension>
          <ns4:PixelYDimension>1424</ns4:PixelYDimension>
          <ns4:FocalLength>353/100</ns4:FocalLength>
          <ns4:GPSAltitudeRef>0</ns4:GPSAltitudeRef>
          <ns4:GPSAltitude>0/1</ns4:GPSAltitude>
          <ns4:GPSMapDatum>WGS-84</ns4:GPSMapDatum>
          <ns4:DateTimeOriginal>2013-02-09T14:47:53</ns4:DateTimeOriginal>
          <ns4:ISOSpeedRatings>
            <rdf:Seq>
              <rdf:li>76</rdf:li>
            </rdf:Seq>
          </ns4:ISOSpeedRatings>
          <ns4:ExifVersion>0220</ns4:ExifVersion>
          <ns4:FlashpixVersion>0100</ns4:FlashpixVersion>
          <ns4:ComponentsConfiguration>
            <rdf:Seq>
              <rdf:li>1</rdf:li>
              <rdf:li>2</rdf:li>
              <rdf:li>3</rdf:li>
              <rdf:li>0</rdf:li>
            </rdf:Seq>
          </ns4:ComponentsConfiguration>
          <ns4:GPSLatitude>42,20.56N</ns4:GPSLatitude>
          <ns4:GPSLongitude>71,5.29W</ns4:GPSLongitude>
          <ns4:GPSTimeStamp>2013-02-09T19:47:53Z</ns4:GPSTimeStamp>
          <ns4:GPSProcessingMethod>NETWORK</ns4:GPSProcessingMethod>
        </rdf:Description>
        <rdf:Description rdf:about="">
          <ns5:DateCreated>2013-02-09T14:47:53</ns5:DateCreated>
        </rdf:Description>
        <rdf:Description rdf:about="">
          <dc:Creator>
            <rdf:Seq>
              <rdf:li>Glymur</rdf:li>
              <rdf:li>Python XMP Toolkit</rdf:li>
```

(continues on next page)

(continued from previous page)

```
            </rdf:Seq>
          </dc:Creator>
        </rdf:Description>
      </rdf:RDF>
    </ns0:xmpmeta>
Contiguous Codestream Box (jp2c) @ (3223, 1132296)
    Main header:
        SOC marker segment @ (3231, 0)
        SIZ marker segment @ (3233, 47)
            Profile:  2
            Reference Grid Height, Width:  (1456 x 2592)
            Vertical, Horizontal Reference Grid Offset:  (0 x 0)
            Reference Tile Height, Width:  (1456 x 2592)
            Vertical, Horizontal Reference Tile Offset:  (0 x 0)
            Bitdepth:  (8, 8, 8)
            Signed:  (False, False, False)
            Vertical, Horizontal Subsampling:  ((1, 1), (1, 1), (1, 1))
        COD marker segment @ (3282, 12)
            Coding style:
                Entropy coder, without partitions
                SOP marker segments:  False
                EPH marker segments:  False
            Coding style parameters:
                Progression order:  LRCP
                Number of layers:  2
                Multiple component transformation usage:  reversible
                Number of resolutions:  2
                Code block height, width:  (64 x 64)
                Wavelet transform:  5-3 reversible
                Precinct size:  default, 2^15 x 2^15
                Code block context:
                    Selective arithmetic coding bypass:  False
                    Reset context probabilities on coding pass boundaries:  False
                    Termination on each coding pass:  False
                    Vertically stripe causal context:  False
                    Predictable termination:  False
                    Segmentation symbols:  False
        QCD marker segment @ (3296, 7)
            Quantization style:  no quantization, 2 guard bits
            Step size:  [(0, 8), (0, 9), (0, 9), (0, 10)]
        CME marker segment @ (3305, 37)
            "Created by OpenJPEG version 2.0.0"
```

That's fairly overwhelming, and perhaps lost in the flood of information is the fact that the codestream metadata is limited to just what's in the main codestream header. You can suppress the codestream and XML details by making use of the `set_option()` function:

```
>>> glymur.set_option('print.codestream', False)
>>> glymur.set_option('print.xml', False)
>>> print(jp2)
File:  nemo.jp2
JPEG 2000 Signature Box (jP  ) @ (0, 12)
    Signature:  0d0a870a
File Type Box (ftyp) @ (12, 20)
    Brand:  jp2
    Compatibility:  ['jp2 ']
```

(continues on next page)

(continued from previous page)

```
JP2 Header Box (jp2h) @ (32, 45)
    Image Header Box (ihdr) @ (40, 22)
        Size:  [1456 2592 3]
        Bitdepth:  8
        Signed:  False
        Compression:  wavelet
        Colorspace Unknown:  False
    Colour Specification Box (colr) @ (62, 15)
        Method:  enumerated colorspace
        Precedence:  0
        Colorspace:  sRGB
UUID Box (uuid) @ (77, 3146)
    UUID:  be7acfcb-97a9-42e8-9c71-999491e3afac (XMP)
Contiguous Codestream Box (jp2c) @ (3223, 1132296)
```

It is possible to easily print the codestream header details as well, i.e.

```
>>> print(j.codestream)    # details not show
```

3.7 . . . add XML metadata?

You can append any number of XML boxes to a JP2 file (not to a raw codestream). Consider the following XML file *data.xml* :

```
<?xml version="1.0"?>
<info>
    <locality>
        <city>Boston</city>
        <snowfall>24.9 inches</snowfall>
    </locality>
    <locality>
        <city>Portland</city>
        <snowfall>31.9 inches</snowfall>
    </locality>
    <locality>
        <city>New York City</city>
        <snowfall>11.4 inches</snowfall>
    </locality>
</info>
```

The `append()` method can add an XML box as shown below:

```
>>> import shutil
>>> import glymur
>>> shutil.copyfile(glymur.data.nemo(), 'myfile.jp2')
>>> jp2 = glymur.Jp2k('myfile.jp2')
>>> xmlbox = glymur.jp2box.XMLBox(filename='data.xml')
>>> jp2.append(xmlbox)
>>> print(jp2)
```

3.8 ... add metadata in a more general fashion?

An existing raw codestream (or JP2 file) can be wrapped (re-wrapped) in a user-defined set of JP2 boxes. To get just a minimal JP2 jacket on the codestream provided by *goodstuff.j2k* (a file consisting of a raw codestream), you can use the `wrap()` method with no box argument:

```
>>> import glymur
>>> glymur.set_option('print.codestream', False)
>>> jp2file = glymur.data.goodstuff()
>>> j2k = glymur.Jp2k(jp2file)
>>> jp2 = j2k.wrap("newfile.jp2")
>>> print(jp2)
File:  newfile.jp2
JPEG 2000 Signature Box (jP  ) @ (0, 12)
    Signature:  0d0a870a
File Type Box (ftyp) @ (12, 20)
    Brand:  jp2
    Compatibility:  ['jp2 ']
JP2 Header Box (jp2h) @ (32, 45)
    Image Header Box (ihdr) @ (40, 22)
        Size:  [800 480 3]
        Bitdepth:  8
        Signed:  False
        Compression:  wavelet
        Colorspace Unknown:  False
    Colour Specification Box (colr) @ (62, 15)
        Method:  enumerated colorspace
        Precedence:  0
        Colorspace:  sRGB
Contiguous Codestream Box (jp2c) @ (77, 115228)
```

The raw codestream was wrapped in a JP2 jacket with four boxes in the outer layer (the signature, file type, JP2 header, and contiguous codestream), with two additional boxes (image header and color specification) contained in the JP2 header superbox.

XML boxes are not in the minimal set of box requirements for the JP2 format, so in order to add an XML box into the mix before the codestream box, we'll need to re-specify all of the boxes. If you already have a JP2 jacket in place, you can just reuse that, though. Take the following example content in an XML file *favorites.xml* :

```
<?xml version="1.0"?>
<favorite_things>
    <category>Light Ale</category>
</favorite_things>
```

In order to add the XML after the JP2 header box, but before the codestream box, the following will work.

```
>>> boxes = jp2.box  # The box attribute is the list of JP2 boxes
>>> xmlbox = glymur.jp2box.XMLBox(filename='favorites.xml')
>>> boxes.insert(3, xmlbox)
>>> jp2_xml = jp2.wrap("newfile_with_xml.jp2", boxes=boxes)
>>> print(jp2_xml)
File:  newfile_with_xml.jp2
JPEG 2000 Signature Box (jP  ) @ (0, 12)
    Signature:  0d0a870a
File Type Box (ftyp) @ (12, 20)
    Brand:  jp2
    Compatibility:  ['jp2 ']
```

(continues on next page)

(continued from previous page)

```
JP2 Header Box (jp2h) @ (32, 45)
    Image Header Box (ihdr) @ (40, 22)
        Size:  [800 480 3]
        Bitdepth:  8
        Signed:  False
        Compression:  wavelet
        Colorspace Unknown:  False
    Colour Specification Box (colr) @ (62, 15)
        Method:  enumerated colorspace
        Precedence:  0
        Colorspace:  sRGB
XML Box (xml ) @ (77, 76)
    <favorite_things>
      <category>Light Ale</category>
    </favorite_things>
Contiguous Codestream Box (jp2c) @ (153, 115236)
```

As to the question of which method you should use, `append()` or `wrap()`, to add metadata, you should keep in mind that `wrap()` produces a new JP2 file, while `append()` modifies an existing file and is currently limited to XML and UUID boxes.

3.9 . . . work with ICC profiles?

A detailed answer is beyond my capabilities. What I can tell you is how to gain access to ICC profiles that JPEG 2000 images may or may not provide for you. If there is an ICC profile, it will be provided in a ColourSpecification box, but only if the `colorspace` attribute is None. Here is an example of how you can access an ICC profile in an example JPX file. Basically what is done is that the raw bytes corresponding to the ICC profile are wrapped in a BytesIO object, which is fed to the most-excellent Pillow package.

```
>>> from glymur import Jp2k
>>> from PIL import ImageCms
>>> from io import BytesIO
>>> # This next step produces a harmless warning that has nothing to do with ICC␣
↪profiles.
>>> j = Jp2k('text_GBR.jp2')
>>> # The 2nd sub box of the 4th box is a ColourSpecification box.
>>> print(j.box[3].box[1].colorspace)
None
>>> b = BytesIO(j.box[3].box[1].icc_profile_data)
>>> icc = ImageCms.ImageCmsProfile(b)
```

To go any further with this, you will want to consult the Pillow documentation.

3.10 . . . create an image with an alpha layer?

OpenJPEG can create JP2 files with more than 3 components (use version 2.1.0+ for this), but by default, any extra components are not described as such. In order to do so, we need to re-wrap such an image in a set of boxes that includes a channel definition box.

This example is based on SciPy example code found at http://scipy-lectures.org/advanced/image_processing/#basic-manipulations . Instead of a circular mask we'll make it an ellipse since the source image isn't square.

```
>>> import numpy as np
>>> import glymur
>>> from glymur import Jp2k
>>> rgb = Jp2k(glymur.data.goodstuff())[:]
>>> lx, ly = rgb.shape[0:2]
>>> X, Y = np.ogrid[0:lx, 0:ly]
>>> mask = ly**2*(X - lx / 2) ** 2 + lx**2*(Y - ly / 2) ** 2 > (lx * ly / 2)**2
>>> alpha = 255 * np.ones((lx, ly, 1), dtype=np.uint8)
>>> alpha[mask] = 0
>>> rgba = np.concatenate((rgb, alpha), axis=2)
>>> jp2 = Jp2k('tmp.jp2', data=rgba)
```

Next we need to specify what types of channels we have. The first three channels are color channels, but we identify the fourth as an alpha channel:

```
>>> from glymur.core import COLOR, OPACITY
>>> ctype = [COLOR, COLOR, COLOR, OPACITY]
```

And finally we have to specify just exactly how each channel is to be interpreted. The color channels are straightforward, they correspond to R-G-B, but the alpha (or opacity) channel in this case is to be applied against the entire image (it is possible to apply an alpha channel to a single color channel, but we aren't doing that).

```
>>> from glymur.core import RED, GREEN, BLUE, WHOLE_IMAGE
>>> asoc = [RED, GREEN, BLUE, WHOLE_IMAGE]
>>> cdef = glymur.jp2box.ChannelDefinitionBox(ctype, asoc)
>>> print(cdef)
Channel Definition Box (cdef) @ (0, 0)
    Channel 0 (color) ==> (1)
    Channel 1 (color) ==> (2)
    Channel 2 (color) ==> (3)
    Channel 3 (opacity) ==> (whole image)
```

It's easiest to take the existing jp2 jacket and just add the channel definition box in the appropriate spot. The channel definition box **must** go into the jp2 header box, and then we can rewrap the image.

```
>>> boxes = jp2.box  # The box attribute is the list of JP2 boxes
>>> boxes[2].box.append(cdef)
>>> jp2_rgba = jp2.wrap("goodstuff_rgba.jp2", boxes=boxes)
```

Here's how the Preview application on the mac shows the RGBA image.


goodstuff_rgba.jp2
Previous
Next
Zoom
Move
Text
Select

3.11 ... work with XMP UUIDs?

Wikipedia states that "The Extensible Metadata Platform (XMP) is an ISO standard, originally created by Adobe Systems Inc., for the creation, processing and interchange of standardized and custom metadata for all kinds of resources."

The example JP2 file shipped with glymur has an XMP UUID.

```
>>> import glymur
>>> j = glymur.Jp2k(glymur.data.nemo())
>>> print(j.box[3]) # formatting added to the XML below
<ns0:xmpmeta xmlns:dc="http://purl.org/dc/elements/1.1/"
             xmlns:ns0="adobe:ns:meta/"
             xmlns:ns2="http://ns.adobe.com/xap/1.0/"
             xmlns:ns3="http://ns.adobe.com/tiff/1.0/"
             xmlns:ns4="http://ns.adobe.com/exif/1.0/"
             xmlns:ns5="http://ns.adobe.com/photoshop/1.0/"
             xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
             ns0:xmptk="Exempi + XMP Core 5.1.2">
      <rdf:RDF>
        <rdf:Description rdf:about="">
          <ns2:CreatorTool>Google</ns2:CreatorTool>
          <ns2:CreateDate>2013-02-09T14:47:53</ns2:CreateDate>
        </rdf:Description>

      .
      .
      .
</ns0:xmpmeta>
```

Since the UUID data in this case is returned as an lxml ElementTree instance, one can use lxml to access the data. For example, to extract the **CreatorTool** attribute value, one could do the following

```
>>> xmp = j.box[3].data
>>> rdf = '{http://www.w3.org/1999/02/22-rdf-syntax-ns#}'
>>> ns2 = '{http://ns.adobe.com/xap/1.0/}'
>>> name = '{0}RDF/{0}Description/{1}CreatorTool'.format(rdf, ns2)
>>> elt = xmp.find(name)
>>> elt
<Element '{http://ns.adobe.com/xap/1.0/#}CreatorTool' at 0xb50684a4>
>>> elt.text
'Google'
```

But that would be painful. A better solution is to install the Python XMP Toolkit (make sure it is at least version 2.0):

```
>>> from libxmp import XMPMeta
>>> from libxmp.consts import XMP_NS_XMP as NS_XAP
>>> meta = XMPMeta()
>>> meta.parse_from_str(j.box[3].raw_data.decode('utf-8'))
>>> meta.get_property(NS_XAP, 'CreatorTool')
'Google'
```

Where the Python XMP Toolkit can really shine, though, is when you are converting an image from another format such as TIFF or JPEG into JPEG 2000. For example, if you were to be converting the TIFF image found at http://photojournal.jpl.nasa.gov/tiff/PIA17145.tif info JPEG 2000:

```
>>> import skimage.io
>>> image = skimage.io.imread('PIA17145.tif')
```

(continues on next page)

(continued from previous page)

```
>>> from glymur import Jp2k
>>> jp2 = Jp2k('PIA17145.jp2', data=image)
```

Next you can extract the XMP metadata.

```
>>> from libxmp import XMPFiles
>>> xf = XMPFiles()
>>> xf.open_file('PIA17145.tif')
>>> xmp = xf.get_xmp()
>>> print(xmp)
<?xpacket begin="" id="W5M0MpCehiHzreSzNTczkc9d"?>
<x:xmpmeta xmlns:x="adobe:ns:meta/" x:xmptk="Exempi + XMP Core 5.1.2">
 <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Description rdf:about=""
    xmlns:tiff="http://ns.adobe.com/tiff/1.0/">
   <tiff:ImageWidth>1016</tiff:ImageWidth>
   <tiff:ImageLength>1016</tiff:ImageLength>
   <tiff:BitsPerSample>
    <rdf:Seq>
     <rdf:li>8</rdf:li>
    </rdf:Seq>
   </tiff:BitsPerSample>
   <tiff:Compression>1</tiff:Compression>
   <tiff:PhotometricInterpretation>1</tiff:PhotometricInterpretation>
   <tiff:SamplesPerPixel>1</tiff:SamplesPerPixel>
   <tiff:PlanarConfiguration>1</tiff:PlanarConfiguration>
   <tiff:ResolutionUnit>2</tiff:ResolutionUnit>
  </rdf:Description>
  <rdf:Description rdf:about=""
    xmlns:dc="http://purl.org/dc/elements/1.1/">
   <dc:description>
    <rdf:Alt>
     <rdf:li xml:lang="x-default">converted PNM file</rdf:li>
    </rdf:Alt>
   </dc:description>
  </rdf:Description>
 </rdf:RDF>
</x:xmpmeta>
<?xpacket end="w"?>
```

If you are familiar with TIFF, you can verify that there's no XMP tag in the TIFF file, but the Python XMP Toolkit takes advantage of the TIFF header structure to populate an XMP packet for you. If you were working with a JPEG file with Exif metadata, that information would be included in the XMP packet as well. Now you can append the XMP packet in a UUIDBox. In order to do this, though, you have to know the UUID that signifies XMP data.:

```
>>> import uuid
>>> xmp_uuid = uuid.UUID('be7acfcb-97a9-42e8-9c71-999491e3afac')
>>> box = glymur.jp2box.UUIDBox(xmp_uuid, str(xmp).encode())
>>> jp2.append(box)
>>> print(jp2.box[-1])
UUID Box (uuid) @ (592316, 1053)
    UUID:  be7acfcb-97a9-42e8-9c71-999491e3afac (XMP)
    UUID Data:
    <ns0:xmpmeta xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:ns0="adobe:ns:meta/
↪" xmlns:ns2="http://ns.adobe.com/tiff/1.0/" xmlns:rdf="http://www.w3.org/1999/02/22-
↪rdf-syntax-ns#" ns0:xmptk="Exempi + XMP Core 5.1.2">
```

(continues on next page)

(continued from previous page)

```
      <rdf:RDF>
        <rdf:Description rdf:about="">
          <ns2:ImageWidth>1016</ns2:ImageWidth>
          <ns2:ImageLength>1016</ns2:ImageLength>
          <ns2:BitsPerSample>
            <rdf:Seq>
              <rdf:li>8</rdf:li>
            </rdf:Seq>
          </ns2:BitsPerSample>
          <ns2:Compression>1</ns2:Compression>
          <ns2:PhotometricInterpretation>1</ns2:PhotometricInterpretation>
          <ns2:SamplesPerPixel>1</ns2:SamplesPerPixel>
          <ns2:PlanarConfiguration>1</ns2:PlanarConfiguration>
          <ns2:ResolutionUnit>2</ns2:ResolutionUnit>
        </rdf:Description>
        <rdf:Description rdf:about="">
          <dc:description>
            <rdf:Alt>
              <rdf:li xml:lang="x-default">converted PNM file</rdf:li>
            </rdf:Alt>
          </dc:description>
        </rdf:Description>
      </rdf:RDF>
    </ns0:xmpmeta>
```

You can also build up XMP metadata from scratch. For instance, if we try to wrap *goodstuff.j2k* again:

```
>>> import glymur
>>> j2kfile = glymur.data.goodstuff()
>>> j2k = glymur.Jp2k(j2kfile)
>>> jp2 = j2k.wrap("goodstuff.jp2")
```

Now build up the metadata piece-by-piece. It would help to have the XMP standard close at hand:

```
>>> from libxmp import XMPMeta
>>> from libxmp.consts import XMP_NS_TIFF as NS_TIFF
>>> from libxmp.consts import XMP_NS_DC as NS_DC
>>> xmp = XMPMeta()
>>> ihdr = jp2.box[2].box[0]
>>> xmp.set_property(NS_TIFF, "ImageWidth", str(ihdr.width))
>>> xmp.set_property(NS_TIFF, "ImageHeight", str(ihdr.height))
>>> xmp.set_property(NS_TIFF, "BitsPerSample", '3')
>>> xmp.set_property(NS_DC, "Title", u'Stürm und Drang')
>>> xmp.set_property(NS_DC, "Creator", 'Glymur')
```

We can then append the XMP in a UUID box just as before:

```
>>> import uuid
>>> xmp_uuid = uuid.UUID('be7acfcb-97a9-42e8-9c71-999491e3afac')
>>> box = glymur.jp2box.UUIDBox(xmp_uuid, str(xmp).encode())
>>> jp2.append(box)
>>> glymur.set_option('print.codestream', False)
>>> print(jp2)
File:  goodstuff.jp2
JPEG 2000 Signature Box (jP  ) @ (0, 12)
    Signature:  0d0a870a
File Type Box (ftyp) @ (12, 20)
```

(continues on next page)

(continued from previous page)

```
    Brand:  jp2
    Compatibility:  ['jp2 ']
JP2 Header Box (jp2h) @ (32, 45)
    Image Header Box (ihdr) @ (40, 22)
        Size:  [800 480 3]
        Bitdepth:  8
        Signed:  False
        Compression:  wavelet
        Colorspace Unknown:  False
    Colour Specification Box (colr) @ (62, 15)
        Method:  enumerated colorspace
        Precedence:  0
        Colorspace:  sRGB
Contiguous Codestream Box (jp2c) @ (77, 115228)
UUID Box (uuid) @ (115305, 671)
    UUID:  be7acfcb-97a9-42e8-9c71-999491e3afac (XMP)
    UUID Data:
    <ns0:xmpmeta xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:ns0="adobe:ns:meta/
↪" xmlns:ns2="http://ns.adobe.com/tiff/1.0/" xmlns:rdf="http://www.w3.org/1999/02/22-
↪rdf-syntax-ns#" ns0:xmptk="Exempi + XMP Core 5.1.2">
      <rdf:RDF>
        <rdf:Description rdf:about="">
          <ns2:ImageWidth>480</ns2:ImageWidth>
          <ns2:ImageHeight>800</ns2:ImageHeight>
          <ns2:BitsPerSample>3</ns2:BitsPerSample>
        </rdf:Description>
        <rdf:Description rdf:about="">
          <dc:Title>Stürm und Drang</dc:Title>
          <dc:Creator>Glymur</dc:Creator>
        </rdf:Description>
      </rdf:RDF>
    </ns0:xmpmeta>
```

CHAPTER 4

"What's new" documents

These document the changes between minor (or major) versions of glymur.

4.1 Changes coming in 0.9

- Python 2.7, 3.5 dropped, support limited to 3.6 and newer.
- OpenJPEG 1.5 dropped, support limited to 2.3.0 and newer.
- lxml and gdal will be required.
- The ColourSpecificationBox instance attribute "icc_profile_data" will change names to "icc_profile", and the old "icc_profile" will change names to "icc_profile_header".

4.2 Changes in glymur 0.8

4.2.1 Changes in 0.8.18

- Fix geotiff UUID corner coordinate string representation.
- Improve parsing warning and error messages.
- Correct improperly raised exception types.
- Remove build/test for Python 3.4 (EOL).
- Fix read-the-docs requirements, improved examples.

4.2.2 Changes in 0.8.17

- Fix parsing of resolution box with negative exponents.

- Add support for ICC profile buffers. The undecoded ICC profile can be accessed via the "icc_profile_data" member of a ColourSpecification box.

4.2.3 Changes in 0.8.16

- Update for Python 3.7.
- Fix documentation bug.

4.2.4 Changes in 0.8.15

- Fix link to readthedocs.
- Fix for invalid progression order display.

4.2.5 Changes in 0.8.14

- Fix bug preventing reads on layers other than the first.

4.2.6 Changes in 0.8.13

- Add support for OpenJPEG threads.

4.2.7 Changes in 0.8.12

- Qualified on OpenJPEG 2.3.0.
- Drop support for Python 3.3.

4.2.8 Changes in 0.8.11

- Qualified on OpenJPEG 2.2.0.

4.2.9 Changes in 0.8.10

- Add pathlib support.

4.2.10 Changes in 0.8.9

- Qualified on Python 3.6.
- Changed travis-ci testing to use Anaconda.

4.2.11 Changes in 0.8.8

- Refactor test suite.
- Fix printing errors in case of bad colr box.
- Fix tests on CentOS when seeing OpenJPEG 1.3

4.2.12 Changes in 0.8.7

- Qualified on OPENJPEG v2.1.2.

4.2.13 Changes in 0.8.6

- State explicit dependence on setuptools.

4.2.14 Changes in 0.8.5

- Relax dependency on lxml; use stdlib ElementTree if necessary.
- Fix bug in XML box processing with certain XML declarations.
- Qualified on OPENJPEG v2.1.1.

4.2.15 Changes in 0.8.4

- Add Anaconda awareness to config module, favor over system package manager.
- Fix issue locating openjpeg dll on windows.

4.2.16 Changes in 0.8.3

- Add gdal interpretation of UUIDBox with GeoTIFF Box specification for JPEG2000 metadata.
- Add support for Python 3.5.
- Add support for Cygwin platform.
- Add write support for UUIDInfo and UUIDList box.
- Relax installation requirement of lxml package from 3.0 to 2.3.2.
- Fix parsing error of bits-per-component box in Python 2.7.

4.2.17 Changes in 0.8.2

- Require at least version 1.5.0 of OpenJPEG.
- Improve read error message when openjpeg library not found.

4.2.18 Changes in 0.8.1

- Add support for bits per component box.

4.2.19 Changes in 0.8.0

- Simplify writing images by moving image data and options into the constructor.
- Deprecate `read()` method in favor of array-style slicing. In order to retain certain functionality, change the following parameters to the `read()` method to top-level properties
  - verbose
  - layer
  - ignore_pclr_cmap_cdef
- Two new properties
  - codestream
  - shape

4.3 Changes in glymur 0.7

4.3.1 Changes in 0.7.3

- added read support back for metadata only when the OpenJPEG library is not installed

4.3.2 Changes in 0.7.2

- added ellipsis support in array-style slicing

4.3.3 Changes in 0.7.1

- fixed release notes regarding Python 3.4

4.3.4 Changes in 0.7.0

- implemented `__getitem__()`, `__setitem__()` support
- added back windows support
- box_id and longname are class attributes now instead of instance attributes

4.4 Changes in glymur 0.6

4.4.1 Changes in 0.6.0

- Added Cinema2K, Cinema4K write support.
- Added irreversible 9-7 transform write support.
- Added set_printoptions, get_printoptions functions.
- Added write support for JP2 UUID, data entry URL, palette, and component mapping boxes.
- Added read/write support for JPX free, number list, and data reference boxes

- Added read support for JPX fragment list and fragment table boxes
- Incompatible change to channel definition box constructor, channel_type and association are no longer keyword arguments
- Incompatible change to palette box constructor, it now takes a 2D numpy array instead of a list of 1D arrays
- Dropped support for 1.3 and 1.4.
- Dropped support for Python 2.6.
- Dropped windows support. It might still work, I don't have access to a windows box with which to test it.
- Added lxml as a package dependency, replacing ElementTree.

4.5 Changes in glymur 0.5

4.5.1 Changes in 0.5.12

- Minor documentation fixes for grammar and style.
- The functions removed in 0.5.11 due to API changes in OpenJPEG 2.1.0 were restored for backwards compatibility. They are deprecated, though, and will be removed in 0.6.0.
  - `glymur.lib.openjp2.stream_create_default_file_stream_v3`
  - `glymur.lib.openjp2.opj.stream_destroy_v3`

4.5.2 Changes in 0.5.11

- Added support for Python 3.4.
- OpenJPEG 1.5.2 and 2.0.1 are officially supported.
- OpenJPEG 2.1.0 is officially supported, but the ABI changes introduced by OpenJPEG 2.1.0 required corresponding changes to glymur's ctypes interface. The functions
  - `glymur.lib.openjp2.stream_create_default_file_stream_v3`
  - `glymur.lib.openjp2.opj.stream_destroy_v3`

  functions were renamed to
  - `glymur.lib.openjp2.stream_create_default_file_stream`
  - `glymur.lib.openjp2.opj.stream_destroy`

  in order to follow OpenJPEG's upstream changes. Unless you were using the svn version of OpenJPEG, you should not be affected by this.

4.5.3 Changes in 0.5.10

- Fixed bad warning issued when an unsupported reader requirement box mask length was encountered.

4.5.4 Changes in 0.5.9

- Fixed bad library load on linux as a result of botched 0.5.8 release. This release was primarily aimed at supporting SunPy.

CHAPTER 5

Known Issues

- Creating a Jp2 file with the irreversible option does not work on windows.
- Eval-ing a `repr()` string does not work on windows.

CHAPTER 6

Roadmap

- Glymur version 0.8.13 will mark the transition of 0.8.x into an LTS series, i.e. bug fixes only with few exceptions.
- Glymur version 0.9.0 will drop support for Python 2.7 and OpenJPEG 1.5.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search