Improved validation and feature extraction for JP2 (JPEG 2000 Part 1) images: the *jpylyzer* tool

Johan van der Knijff ^{1,2}***, René van der Ark*** ^{1}***, Carl Wilson*** ^{3}

^{1}***KB/ National Library of the Netherlands, The Hague, the Netherlands;*** ^{2}***Open Planets Foundation, Boston Spa, UK;*** ^{3}***The British Library, Boston Spa, UK***

Abstract

This paper presents 'jpylyzer', a dedicated software tool for validation and feature extraction of JP2 (JPEG 2000 Part 1) images. We first discuss the importance of format validation, and show that for the JP2 format, current tools have deficiencies that make their use in a preservation context problematic. We describe the main characteristics of the new jpylyzer software, and provide some case studies that illustrate its utility in operational imaging workflows.

Introduction

JP2 (or, JPEG 2000 Part 1) is becoming increasingly popular in the cultural heritage sector as a format for access and long-term archival storage of digital image data. Examples include (but are not limited to) the National Library of the Netherlands [1], the British Library [2], the Wellcome Library [3], Library of Congress [4], the National Library of Norway [5], and the National Library of the Czech Republic [6]. These institutions are using JP2 as a master format for mass digitization projects, and some have also started migrating their existing image archives (which are typically in TIFF format) to JP2.

Both digitization and migration involve imaging workflows in which quality assurance is an essential component. One important aspect of this is a check that ensures that the created JP2s are compliant with the format's specification, which is defined by Part 1 of the JPEG 2000 standard [7]. Also, JPEG 2000 leaves one with a wide range of encoding options, which can be optimized to the envisaged use of the images. These options are typically defined using institute-specific profiles. The verification of these options is thus also an important aspect of the quality assurance component.

Previous work has shown that some encoders produce JP2s that contain (often subtle) deviations from the standard. Some encoders generate files that are not even JP2s at all, but instead JPX (JPEG 2000 Part 2) images (which, in some cases, can be virtually indistinguishable from JP2) [8],[9]. Corrupted images that are the result of hardware failure (e.g. brief network interruptions or malfunctioning hard disks) are another concern.

Importantly, neither non-compliance to the standard nor corruption automatically implies that an image cannot be displayed in a viewer. For instance, any JP2 viewer will be able to display most JPX images. However, such images may contain embedded ICC profiles that the viewer will simply ignore, which means that they will be lost if such tools are used to migrate the image to some other format. In addition, a JPEG 2000 image typically contains several approximation scales (e.g. quality layers, resolution levels), and often the lower level approximations can be displayed without loading the image as a whole. In combination with JPEG 2000's error resilience features, this means that viewers may render a damaged JP2 without complaining.

Compliance with the format's specification can be tested using format validator tools. Ideally such tools should be able to detect most forms of file corruption as well. As no sufficiently thorough validator exists for JP2, in this paper we present a software tool that is aimed at bridging this gap. The *jpylyzer* tool is a strict, thorough JP2 validator. It also performs comprehensive feature extraction, which can serve as input for the verification of images against institute-specific encoding profiles.

Outline of this paper

The remainder of this paper is organized as follows. First we give an overview of the general structure of the JP2 format, illustrating the fundamentals of validation in the process. Then we summarize the current state of the art of JP2 validation and feature extraction, and from this we argue that the current situation justifies the development of a new, better tool. We then outline the main philosophy and features of *jpylyzer*, including a note on its envisaged role in quality assurance workflows. We also cover the tool's inherent limitations. Next we move on to a number of case studies that demonstrate the utility of *jpylyzer* in a variety of practical settings, including operational imaging workflows. Finally, we outline some envisaged future directions of *jpylyzer*'s development.

Structure of JP2 and basis for validation

At the top level, a JP2 file is made up of a collection of building blocks known as 'boxes'. Some boxes ('superboxes') are containers for other boxes. **Figure 1** gives an overview of the top-level boxes in a JP2 file. Some of these are required, whereas others (indicated with dashed lines in the Figure) are optional. The order in which the boxes appear in the file is subject to some constraints. For example, the first box in a JP2 must always be a 'Signature' box, followed by a 'File Type' box. Some boxes may have multiple instances (e.g. the 'Contiguous Codestream' box), whereas others must be unique (e.g. the 'JP2 Header' box). These constraints, which are all precisely defined by the format's specification, provide a first set of criteria that a validator should check at the highest level. For convenience we will refer to this as 'level 1 validation' in the remainder of this paper.

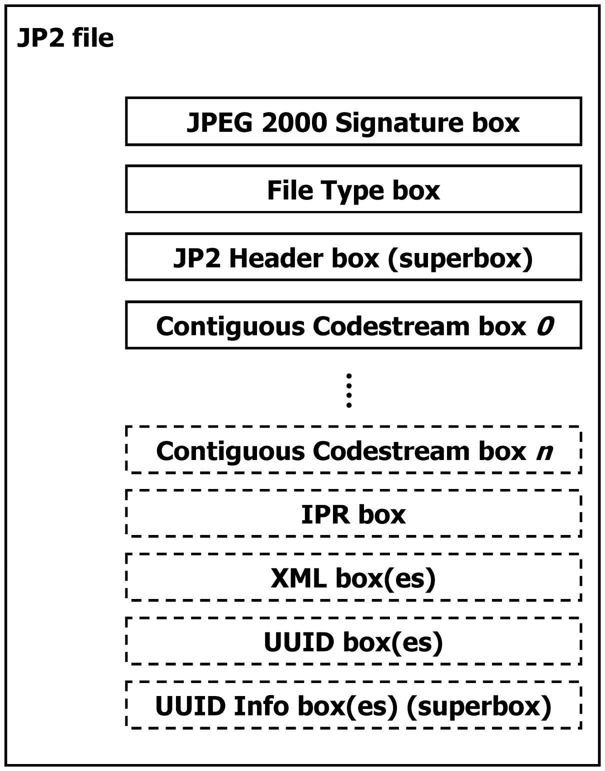

JP2 file
JPEG 2000 Signature box
File Type box
JP2 Header box (superbox)
Contiguous Codestream box 0
Contiguous Codestream box n
IPR box
XML box(es)
UUID box(es)
UUID Info box(es) (superbox)

Figure 1. *Top-level view of a JP2 file. Boxes with dashed borders are optional. 'Superbox' denotes a box that contains other box(es).*

All boxes are defined by a generic binary structure, which comprises 1) a fixed-length 'box length' field that indicates the total size of the box (in bytes); 2) a fixed-length 'box type' field that specifies the type of information that can be found in this box; and 3) the actual contents of the box (in case of a 'superbox' this will hold its child boxes, which can be parsed recursively). In some cases a box will also contain an 'extended box length' field, which is needed if the size of a box exceeds the maximum value that can be stored in the 4-byte 'box length' field.

Again, the format specification provides a detailed description of the contents of each box, its included fields, and the allowed values for each field. For example, the 'Colour Specification box' (which is a child of the 'JP2 Header box') contains a field that specifies how the colour space of an image is defined. Its value must either be 1 ('enumerated colourspace') or 2 ('restricted ICC profile'). Such constraints provide a lower level set of criteria that should be included in the validation process. We will call this 'level 2 validation'.

Finally, there are several instances where the information in one box must be consistent with another box. As an example, most of the fields in the 'Image Header box' (again a child of the 'JP2 Header box') are redundant with information in the 'Contiguous Codestream box'. Validation should include checks to establish the consistency between the two boxes ('level 3 validation').

Tools for JP2 validation

To the best of our knowledge, the only tool that is currently available for validating JP2 is JHOVE (**J**STOR/**H**arvard **O**bject **V**alidation **E**nvironment) [10]. It provides validation functionality for 12 file format classes, and includes a JPEG 2000 module which handles both Part 1 (JP2) and Part 2 (JPX) images. JHOVE's documentation reveals that most of its validation involves a relatively shallow check of the general box structure ('level 1 validation'), whereas only a very limited number of individual fields are validated ('level 2'). Importantly, it does not validate the contents of the image codestream, and the documentation does not mention any consistency checks ('level 3').

We conducted a number of experiments in which we deliberately removed codestream data from a JP2. In one of these tests we trimmed a 2 MB file down to only 4 kilobytes; according to JHOVE 1.6 this file was 'well-formed and valid' JP2. We also ran a test on images that were created using Adobe's JPEG 2000 plugin (version: 2.0, 2007) that comes with Photoshop CS4. This plugin produces files that contain features from JPEG 2000 Part 2 (JPX), and which are not permitted in JP2. Both Parts 1 and 2 of the standard use the 'brand' field in the 'File Type box' to identify whether a file is JP2 or JPX. However, Adobe's plugin erroneously uses the JP2-specific value, even though the files are technically JPX. According to JHOVE however these files are also 'well-formed and valid' JP2. In another test we changed the image height field in the 'Image Header box', making it inconsistent with the codestream. Again, this wasn't picked up by JHOVE.

Summarizing, JHOVE's validation functionality is rather limited, and the tool will often judge files that contain major defects with respect to the format specification to be 'well-formed and valid' JP2.

Tools for JP2 feature extraction

ExifTool

ExifTool [11] is an open source command-line tool for reading, writing and editing meta information for a wide variety of file formats. It does also support JP2; however, it does not extract any codestream information, such as progression order, number of quality layers, wavelet transformation type (lossless vs lossy), and so on.

Kakadu

Kakadu's 'kdu_expand' tool can be used to extract detailed codestream information (using its –record switch), but its output does not include any information from the other header boxes [12]. Besides, the software is released under a somewhat restricted license.

JHOVE

JHOVE [10] provides detailed information from both JP2's header boxes and the codestream. Its output can be difficult to interpret because most properties are reported as literal representations of the header fields. As an example to illustrate this, JHOVE's JP2 output includes the reportable 'ProgressionOrder', which is represented as an integer number. These numbers correspond to descriptive values. For instance, a value '2' denotes 'RPCL' progression order. JHOVE only provides the number here, with no reference to its interpretation.

The same applies to most other codestream properties, which makes their interpretation rather challenging (especially since the codestream specification is not freely available).

ImageMagick

ImageMagick [13] includes an 'identify' tool that extracts meta-information for many image formats. It only provides very limited information for JP2 images. This appears to be largely related to its dependency for JP2 on the open source JasPer library [14], which does not support all features of the format (e.g. the 'Resolution box' is not even supported at all). Besides, the library has serious performance issues (both in terms of speed and stability), meaning that ImageMagick is not very useful within this particular context.

Roundup

Summarizing, a number of tools are available for feature extraction. However, these tools either provide partial coverage of JP2's feature set, or produce output that –to most users- is difficult to interpret.

Jpylyzer: general philosophy and features

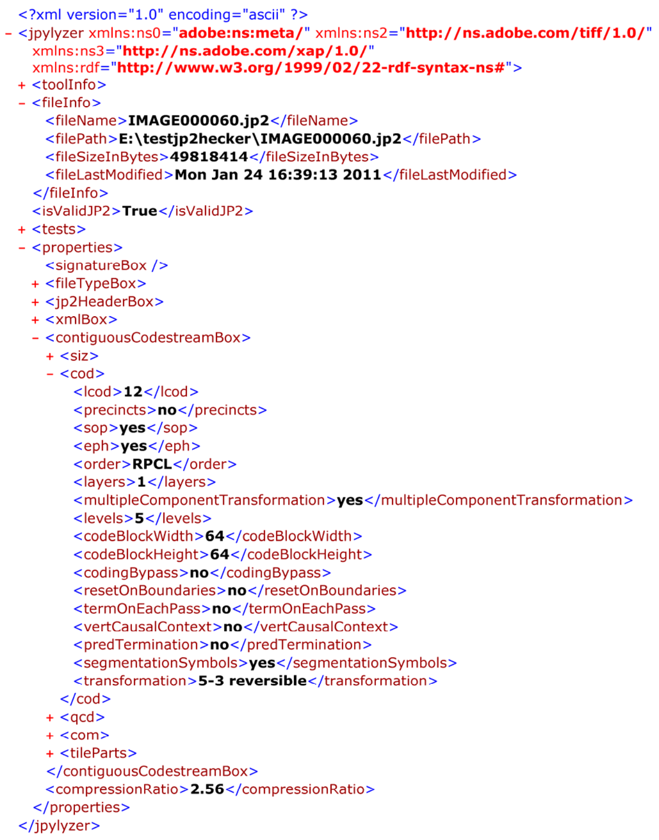
The lack of any reliable tool for JP2 validation prompted us to the development of a new tool called *jpylyzer*. The overall objective was to develop a strict validator that adheres closely to the format specification. Its main use cases are to:

- verify whether an encoder produces standard-compliant JP2s;
- detect JP2s that are corrupted (e.g. images that are truncated or have missing data).

Since such an exhaustive validation implies that (nearly) all boxes and header fields are read and analyzed, it was obvious to include functionality for reporting these properties, making *jpylyzer* also an elaborate feature extractor. Unlike JHOVE, *jpylyzer*'s scope is restricted to the JP2 format only (although its feature extraction functionality will work with JPX for features that are shared with JP2). It also extracts meta-information from ICC profiles that may be embedded in a JP2. In addition, the validation functionality includes checks that verify the integrity and completeness of image codestreams.

The main philosophy behind *jpylyzer* was to create a tool that is lightweight, simple and performant. In a nutshell, *jpylyzer*'s validation process is defined as a series of tests that either return 'True' or 'False'. A file is deemed 'valid' JP2 if it passes every single test, and 'not valid' otherwise. The tests include a verification of the general box structure ('level 1'), tests on the validity of individual fields ('level 2') and a number of consistency checks ('level 3'). As we wanted to make things as transparent to the user as possible, the software is accompanied by a comprehensive User Manual, [15] that documents every single test that is part of the validation process, as well as every extracted feature. The manual also provides more general background information on JP2 and validation.

Figure 2 gives an example of *jpylyzer*'s output, which is reported as XML. From top to bottom, the Figure shows some general file information, the outcome of the validation process ('isValidJP2', which is either 'True' or 'False'), the properties of the codestream's 'coding style default' header, and the calculated compression ratio. Note that the Figure only shows part of the output, and that the results of the individual tests and most other properties are collapsed (i.e. hidden) here, although they are included in the output.



```
  <?xml version="1.0" encoding="ascii" ?>
- <jpylyzer xmlns:ns0="adobe:ns:meta/" xmlns:ns2="http://ns.adobe.com/tiff/1.0/"
    xmlns:ns3="http://ns.adobe.com/xap/1.0/"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  + <toolInfo>
  - <fileInfo>
      <fileName>IMAGE000060.jp2</fileName>
      <filePath>E:\testjp2hecker\IMAGE000060.jp2</filePath>
      <fileSizeInBytes>49818414</fileSizeInBytes>
      <fileLastModified>Mon Jan 24 16:39:13 2011</fileLastModified>
    </fileInfo>
    <isValidJP2>True</isValidJP2>
  + <tests>
  - <properties>
      <signatureBox />
    + <fileTypeBox>
    + <jp2HeaderBox>
    + <xmlBox>
    - <contiguousCodestreamBox>
      + <siz>
      - <cod>
          <lcod>12</lcod>
          <precincts>no</precincts>
          <sop>yes</sop>
          <eph>yes</eph>
          <order>RPCL</order>
          <layers>1</layers>
          <multipleComponentTransformation>yes</multipleComponentTransformation>
          <levels>5</levels>
          <codeBlockWidth>64</codeBlockWidth>
          <codeBlockHeight>64</codeBlockHeight>
          <codingBypass>no</codingBypass>
          <resetOnBoundaries>no</resetOnBoundaries>
          <termOnEachPass>no</termOnEachPass>
          <vertCausalContext>no</vertCausalContext>
          <predTermination>no</predTermination>
          <segmentationSymbols>yes</segmentationSymbols>
          <transformation>5-3 reversible</transformation>
        </cod>
      + <qcd>
      + <com>
      + <tileParts>
      </contiguousCodestreamBox>
      <compressionRatio>2.56</compressionRatio>
    </properties>
  </jpylyzer>
```

Figure 2. *Example of jpylyzer output, see main text for explanation.*

Robustness is particularly important: files that are damaged, malformed or that even are of a completely different format altogether, are handled gracefully, and should never result in crashes.

Jpylyzer is written in Python. To ensure sustainability of the code over time while simultaneously making it widely supported on current systems, we created code that is fully compatible with both Python 2.7 and Python 3.2 (and more recent). It is released under a permissive (GNU Lesser) open source license, and we took particular care in producing understandable, heavily commented source code. Acknowledging that not all users may wish to install Python on their systems, we also provide self-contained binary packages for Windows.

Limitations of jpylyzer

Even though *jpylyzer*'s analysis of JP2 is very comprehensive, the tool should not be seen as a 'one stop' solution. First of all, it is important to stress that if a file passes all tests, this should be seen as an indication that it is *probably* valid JP2. This (intentionally) implies a certain degree of remaining uncertainty, which is related to the following. First of all, *jpylyzer* (or any other format validator for that matter) 'validates' a file by trying to prove that it does *not* conform to the standard. It cannot prove that that a file *does* conform to the standard. In addition, *jpylyzer* is not capable of 'validating' the actual image data (i.e. the compressed bitstream segments). Doing so would require decoding the image as a whole, and this is completely out of *jpylyzer*'s scope. In the case of a TIFF to JP2 imaging workflow, this could imply a quality assurance component that includes a check by *jpylyzer*, accompanied by a rendering test or a pixel-wise comparison of the source and destination images.

In the following sections we present four case studies that serve to illustrate the utility of *jpylyzer*.

Detection of damaged JP2s at British Library

The JISC 1 Newspaper Collection

JISC 1 was a project funded by the Joint Information System Comittee which digitized 275,000 19th Century newspaper issues held by The British Library (BL). Digitization was performed by an external partner, who delivered:

- a master 8-bit greyscale TIFF image for each of the 2 million pages
- a service monochrome TIFF for each page
- a service TIFF for each article
- an XML metadata file for each page and article

Conversion to JP2

The 2 million TIFF master images comprised 80 TB in total, and were to be ingested into The British Library's Digital Library System (DLS), which creates 4 copies of each item in order to protect against corruption and disaster. In order to save money on storage costs for the master images, it was decided to convert the 8-bit TIFFs to the JP2 format. This was an emerging preservation format, and reduced the size of the master image collection to 45 TB, yielding a total storage saving of 140 TB (4 x 35 TB). This conversion was implemented as an ingest workflow that converted each master TIFF to a JP2, and then used JHOVE 1.6 to validate the converted image before ingest into DLS.

Post ingest issues

The ingest of 2 million images was not without errors. While investigating some of the issues that had had ingest problems, it was noticed that some "noisy" images had been produced. In spite of this, JHOVE had indicated that these files were valid. This raised the possibility that some percentage of the ingested master images were not valid JP2 files. Examination of the ingested master images revealed some very small files which were corrupt, but some of the original examples were of the expected size and were also broken. It was not clear if there was a suitable tool for validating the images held in DLS.

Analysis of BL JP2 Collections

Early in 2012 *jpylyzer* was tested against the broken image sample set, and it identified every broken image. It was then used to validate every single master image in the collection. This was performed as a single thread on a 3 year old Xeon server. Performance testing suggested this should complete in less than a month.

The validation completed in 3 weeks, and found 676 invalid JP2s in the entire collection, an error rate of 0.03%. The TIFF originals for the collection are still held on an intermediate store, and can be used to re-generate the broken JP2s.

Following this, *jpylyzer* was used to validate the British Library's JISC 2 newspaper collection, as well as its 19th Century book collection. This revealed a further 3 invalid JP2s for JISC2, and none for the 19th Century books. The table below shows some performance statistics for these analyses. These figures include the time that was needed to unpack the images from a ZIP container, so the actual times for the *jpylyzer* analyses (which were not recorded separately) are significantly less.

Performance statistics *jpylyzer* analysis of BL JP2 collections

Collection	JISC 1 newspapers	JISC 2 newspapers	19th Cent. books
# of images	2,152,116	1,161,210	22,507,396
Total size (TB)	45	25	15
Av. image size (MB)	21.8	22.7	0.7
# of threads	1	1	3
Time (days)	21	11	21
Images /day / thread	100,000	100,000	300,000
TB / day / thread	2	2	0.25

Metamorfoze migration at KB

'Metamorfoze' is the Netherlands' national programme for the preservation of paper heritage. It is a collaborative effort of the National Library of the Netherlands (KB) and the National Archives of the Netherlands [16]. It employs digitization as one of its conservation methods (preservation imaging). Thus far, Metamorfoze has been using uncompressed TIFF as its preservation format. This has resulted in about 90 TB worth of TIFF images being stored at the KB by October 2011. With a further 56 TB arriving over 2012, this will result in 146 TB by the end of 2012.

The KB is preparing to migrate these images to lossless JP2, as this would result in a significant reduction in storage costs. One particular risk of such a large scale migration is that hardware failure may result in corrupted images. Since some of the Metamorfoze material is irreplaceable (because the paper originals are in poor shape), it is vital to have a workflow that includes checks that ensure the integrity of the migrated images.

To this end, the Metamorfoze migration workflow will use *jpylyzer* to verify that each created image is valid and intact JP2. In addition, *jpylyzer*'s feature extraction output is used to check that each image's encoding options (compression type, number of decomposition levels and quality layers, progression order, tile- and codeblock size, error resilience markers) match a pre-defined profile. Finally, a pixel-wise comparison is done between each JP2 and its source TIFF image. Images that fail any of these tests are added to an error log, and the associated batches will not pass the quality control. Interestingly, some initial tests of the migration workflow yielded images that did not pass *jpylyzer*'s validation, with *jpylyzer* reporting an incomplete image codestream. The problem could be traced down to a software-related error that occurred while copying files over a network connection. This resulted in files that had some trailing kilobytes missing. Even though this particular problem is unlikely to occur in an operational production environment, it does demonstrate *jpylyzer*'s effectiveness in detecting such errors.

Quality control at Wellcome Library

The Wellcome Library is preparing to use *jpylyzer* for validating JP2 images, both those that are produced internally as well as those received from external suppliers. They will also use the tool's output to verify that the technical characteristics of the images match a profile that defines aspects such as compression ratio, progression order and the number of quality layers. To this end, the profile is coded as an XML schema. This reduces the verification process to validating *jpylyzer*'s output file for each image against this schema [17].

Detection of non-compliant images

In our discussion of JHOVE we already mentioned the issue of the problematic JPX files that are produced by Adobe Photoshop. These files are easily mistaken for JP2; however, validating these files with *jpylyzer* will confirm that they are not valid JP2.

Conclusion and future directions

The development of *jpylyzer* was triggered by a lack of tools that are able to check the integrity and validity of JP2 files, a format that is becoming increasingly important in archival and preservation imaging. The first experiences described in this paper demonstrate the potential role and value of this tool in a variety of imaging workflows.

We consider the software to be in beta stage at this point. Further testing is needed to ensure its stability under as wide a range of potential image malformations as possible. Also, several improvements that will make the validation of image codestreams even more thorough are under way.

At the time of writing, efforts are ongoing to create Debian packages for a range of Linux-based architectures [18], [19]. This should simplify the installation process, remove any dependencies on Python, and ultimately we hope that this will contribute to the further adoption of the tool.

Finally, *jpylyzer* is hosted by the Open Planets Foundation. This ensures the involvement of a wider community in its maintenance and further development. This is an important safeguard towards the long-term support and sustainability of the software.

Acknowledgements

Christy Henshaw (Wellcome Library), Ross Spencer (The National Archives), Wouter Kool, Reinier Deinum (both KB) are all thanked for providing valuable feedback, some of which has served as input to this paper. Dave Tarrant (University of Southampton/Open Planets Foundation), Miguel Ferreira, Rui Castro, Hélder Silva (KEEP Solutions) and Rainer Schmidt (Austrian Institute of Technology) are thanked for initiating the work on Debian packages for *jpylyzer* (which is ongoing at the time of writing). Lars Buitinck (University of Amsterdam) is also thanked for contributing to the code.

Jpylyzer downloads

For more information on *jpylyzer*, including source code, binaries and documentation, please follow the links below.

Open Planets Foundation

http://www.openplanetsfoundation.org/software/jpylyzer

Github repository

https://github.com/openplanets/jpylyzer/

Funding

This work was partially supported by the SCAPE Project. The SCAPE project is co-funded by the European Union under FP7 ICT-2009.4.1 (Grant Agreement number 270137).

References

[1] R. Gillesse, J. Rog & A. Verheusen, Alternative File Formats for Storing Master Images of Digitisation Projects. Koninklijke Bibliotheek, Den Haag (2008). http://www.kb.nl/hrd/dd/dd_links_en_publicaties/publicaties/Alternative_File_Formats_for_Storing_Masters_2_1.pdf.

[2] R. McLeod, & P. Wheatley, Preservation Plan for Microsoft — Update Digital Preservation Team. The British Library, London (2007). http://www.bl.uk/aboutus/stratpolprog/ccare/introduction/digital/digpresmicro.pdf

[3] C. Henshaw, We need how much storage? Wellcome Library, London (2010). http://jpeg2000wellcomelibrary.blogspot.com/2010/06/we-need-how-much-storage.html

[4] R. Buckley, & R. Sam, JPEG 2000 Profile for the National Digital Newspaper Program. Library of Congress, Washington (2006). http://www.loc.gov/ndnp/guidelines/docs/NDNP_JP2HistNewsProfile.pdf

[5] National Library of Norway, Digitization of books in the National Library — methodology and lessons learned. National Library of Norway, Oslo (2007). http://www.nb.no/content/download/2326/18198/version/1/file/digitizing-books_sep07.pdf

[6] B. Vychodil, JPEG2000 - Specifications for The National Library of the Czech Republic. Seminar JPEG 2000 for the Practitioner, Wellcome Trust, London (2010). http://www.dpconline.org/component/docman/doc_download/520-jp2knov2010bedrich

[7] ISO/IEC, Information technology — JPEG 2000 image coding system: Core coding system. ISO/IEC 15444-1, Second edition. ISO/IEC, Geneva (2004). http://www.jpeg.org/public/15444-1annexi.pdf ("Annex I: JP2 file format syntax" only)

[8] ISO/IEC, Information technology - JPEG 2000 image coding system: Extensions. ISO/IEC 15444-2, First edition. ISO/IEC, Geneva (2004). http://www.jpeg.org/public/15444-2annexm.pdf ("Annex M: JPX extended file format syntax" only)

[9] J. van der Knijff, JPEG 2000 for Long-term Preservation: JP2 as a Preservation Format. D-Lib 17, 5/6 (2011). http://www.dlib.org/dlib/may11/vanderknijff/05vanderknijff.html

[10] JHOVE - JSTOR/Harvard Object Validation Environment. http://hul.harvard.edu/jhove

[11] P. Harvey, ExifTool. http://www.sno.phy.queensu.ca/~phil/exiftool/

[12] D. Taubman, Kakadu. http://www.kakadusoftware.com/

[13] ImageMagick. http://www.imagemagick.org

[14] M. Adams, JasPer. http://www.ece.uvic.ca/~frodo/jasper/

[15] J. van der Knijff, Jpylyzer: validator and properties extractor for JPEG 2000 Part 1 (JP2), User Manual. KB/National Library of the Netherlands / Open Planets Foundation. Latest version available from: https://github.com/openplanets/jpylyzer/downloads

[16] Metamorfoze. http://www.metamorfoze.nl/programme
[17] C. Henshaw, pers. comm.
[18] M. Ferreira, Sustainability and adoption of preservation tools. Open Planets Foundation (2012). http://www.openplanetsfoundation.org/blogs/2012-02-15-sustainability-and-adoption-preservation-tools
[19] D. Tarrant, Turning GitHub Code into Debian Packages - The OPF Way. Open Planets Foundation (2012). http://www.openplanetsfoundation.org/blogs/2012-03-08-turning-github-code-debian-packages-opf-way

Author Biography

Johan van der Knijff holds an MSc in Physical Geography from Utrecht University (1998). He previously worked on the development of hydrological simulation models. His current job as a digital preservation researcher at the KB focuses on preservation-related aspects of file formats. He is also liaison to JPEG on behalf on the Open Planets Foundation.

René van der Ark received his MA in Information Science from the University of Groningen in 2008, specializing in comparative statistics and spatial linguistics. Since 2009 he has been active as a research programmer in the employ of the KB (National Library of the Netherlands).

Carl Wilson has worked as a software developer and requirements analyst for 20 years, 14 of them at The British Library. He has worked on digital preservation projects since 2003, including the PLANETS, and SCAPE projects, particularly focusing on the identification and validation of preservation file formats.