Jpylyzer: Analysing JP2000 files with a community supported tool

David Tarrant
School of Electronics and Computer Science
University of Southampton
Southampton
UK
davetaz@ecs.soton.ac.uk

Johan Van Der Knijff
National Library of the Netherlands
Prins Willem-Alexanderhof 5
2595 BE
Den Haag
johan.vanderknijff@kb.nl

This paper introduces Jpylyzer, a validator and feature extractor for JP2 images. This simple tool is designed to do one task, validation, reliably and in a way conforming strictly to the JPEG 2000 Part 1 (ISO/IEC 15444-1) specification. In order to validate JP2 files, Jpylyzer examines the features used for conformance, thus consequently is able to list the features used. Further, this paper outlines how the development of Jpylyzer was strictly controlled in order to make Jpylyzer a tool that can be adopted by a wider community. We outline the software "curation" process being adopted within the OPF and SCAPE digital preservation projects. This process has resulted in Jpylyzer being successfully adopted into the debian/Ubuntu community and prepared Jpylyzer for submission to other platform "stores" and download sites.

Jpylyzer was specifically created to answer the following questions that you might have about any JP2 file:

1. Is this really a JP2 and does it really conform to the format's specifications (validation)?
2. What are the technical characteristics of this image (feature extraction)?

At the highest level, a JP2 file is made up of a collection of boxes. A box can be thought of as the fundamental building block of the format. Some boxes ('superboxes') are containers for other boxes. Figure 1 gives an overview of the topâĂŘlevel boxes in a JP2 file.

The structure of a JP2 file is important, as is the order of the boxes, e.g. the first box in a JP2 file must always be a 'Signature' box, followed by a 'File Type' box. Additionally some boxes allow multiple instances (e.g. 'Contiguous Codestream' box), whereas others (e.g. 'JP2 Header' box) must be unique. Each box must also have the same four part structure, as shown by Figure 2. The strict order and properties these boxes allows for easy validation.

Jpylyzer parses a JP2 file, based on the JP2 format specification (ISO/IEC 15444-1) [1]. It then subjects the file's contents to a large number of tests, each of which is based on the requirements and restrictions that are defined by the standard. If a file fails one or more tests, this implies that it does not conform to the standard, and is not valid JP2. Importantly, this presumes that Jpylyzer's tests accurately reflect the format specification, without producing false positives.

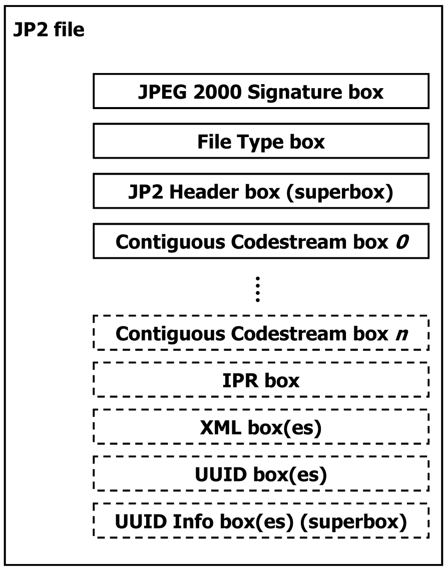
JP2 file
JPEG 2000 Signature box
File Type box
JP2 Header box (superbox)
Contiguous Codestream box 0
Contiguous Codestream box n
IPR box
XML box(es)
UUID box(es)
UUID Info box(es) (superbox)

Figure 1: Top-level overview of a JP2 file (boxes with dashed borders are optional)

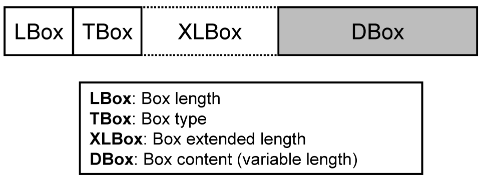
LBox
TBox
XLBox
DBox
LBox: Box length
TBox: Box type
XLBox: Box extended length
DBox: Box content (variable length)

Figure 2: JP2000 box structure

It is important to note that Jpylyzer does not check that a JP2 will render; it does not examine the bitstream content of the file. For scalability, and the fact that other render checking tools exist [3] [2] [4], it has been decided that this functionality will never be added to the tool.

Further Jpylyzer adopts a closed world assumption. Jpylyzer validates a file by trying to prove that is does not conform to the standard. Additionally due to ambiguities in the specification (that have been reported by the creators of Jpylyzer), not all aspects of the standard can currently be checked for validity. It is hoped in the future that these issues will be resolved by the JP2000 standardization community.

Jpylyzer was designed for purpose. The British Library has for some time been digitizing content into the JP2 format, however to this point they had no reliable and scalable way to validate these files. Through limited manual inspection, a series of corrupt files were located, leading to question about just how many files were corrupt. Jpylyzer was designed to suit such a purpose, being a simple command line tool that quickly analyses files and gives simplistic (as well as extended) output. Using Jpylyzer, the British library was able to successfully analyse 2.15 million JP2000 files and discovered 676 invalid images. This process took 21 days using a single threaded process.

From an early stage there was a strong demand for Jpylyzer to exist as a cross platform tool. This demand came from the user community, the most important community to drive the adoption of Jpylyzer outside of the relatively small digital preservation community.

Written in python, Jpylyzer provides a simple command line interface to which a user invokes Jpylyzer along with the location of one or more JP2 files. Reports are then output in XML format and the user is able to save these to a file using simple pipelines. Additionally the Jpylyzer library can be imported into other python systems and utilized as a library. Importantly, Jpylyzer can also be compiled into "native" C code. As well as speeding up the execution of Jpylyzer, this also allows Jpylyzer to be distributed for platforms that don't have python support natively.

The Jpylyzer XML output is split into several sections as detailed in Figure 3, each equally important for effective preservation of JP2 files. The first two sections (toolInfo and fileInfo) detail the provenance information relating to the version of Jpylyzer (the tool) used and the details of the file scanned. The remaining three elements detail the validation results, beginning by simply stating whether the file is valid or not before detailing the results of each test and properties of each box.

So at this stage, Jpylyzer provides a small, focused and contained tool for assisting in the digital preservation of JP2 files. It also conforms strictly to the JP2000 Part 1 specification and only utilizes off the shelf python libraries that con be compiled for native use. These aspect forms the basis of the OPF/Scape software curation process (Figure 4).

The first two steps usually represent as far as a prototypical or research based tool gets. To progress beyond step two, requires the adoption of good software development practices. Beyond using a version control system, good practice involves adopting a version numbering system, relating this to a ticketing system and actually considering what your commit messages should be! Again without good practice,

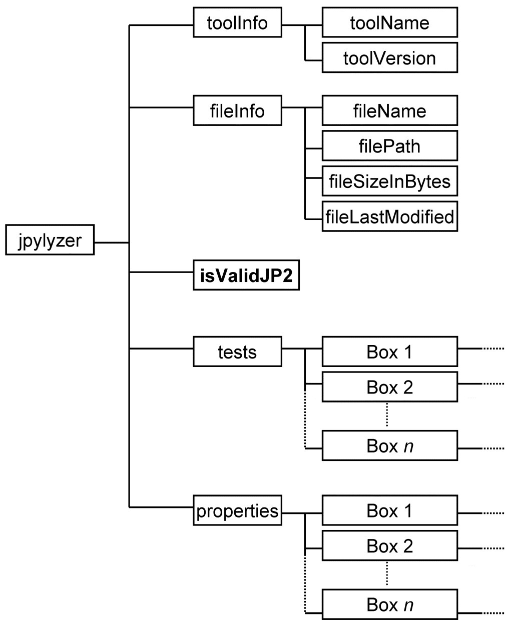

Figure 3: Jpylyzer's XML output structure

it is likely that the development of the tool will stall later in the process.

Step 4 relates to the packaging of the tool ready for distribution, not as a source code zip file, but as a one-click install package. Beyond this your package needs documentation, including user documentation, man page and well constructed changelog. Your package will also need to conform to all the strict packaging specifications applied by the platform developers (e.g. Apple, Microsoft and Linux) to prevent your app being rejected or simply not working due to specific requirements.

These stages sound easy, but if the tool has not been carefully curated they are not. If however, a piece of software is curated properly, then there is a chance it might not only be accepted into the "App Store" but also gain interest from the community in using and improving your tool.

Jpylyzer represents and exemplar in the preservation community of this process working. In May 2011, Jpylyzer found itself a mentor in the debian/Ubuntu community (a requirement for adoption). Through continued work with this mentor and the wider community, it is hoped that Jpylyzer will soon be readily available to a much bigger community via traditional methods. Further, through listening and working with this community, it is hoped that a much larger number of developers can support this tool well into the future.

Jpylyzer, a digital preservation software sustainability success story.

Thanks go to the Dutch National Library, OPF and SCAPE projects. Also to the debain mentor community for willing to adopt the tool.

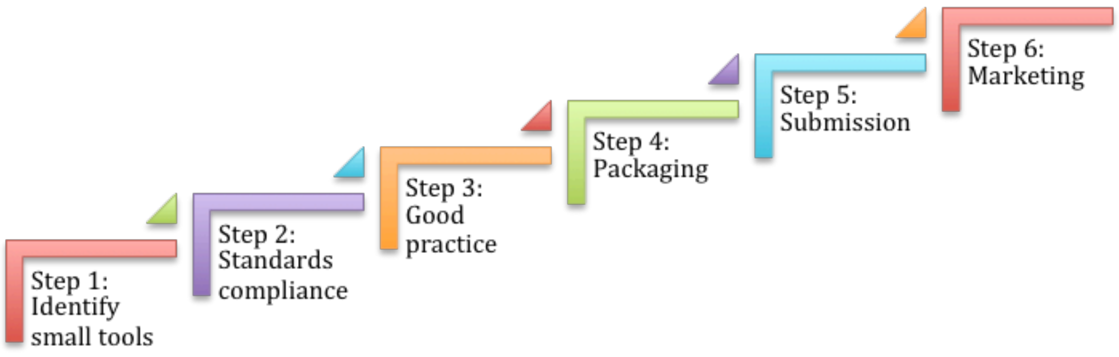

Figure 4: The Software Curation Process

1. REFERENCES

[1] Jpeg 2000 image coding system: Core coding system. *Information technology, ISO/IEC 15444-1:2004*, 2004.

[2] O. Chum, J. Philbin, M. Isard, and A. Zisserman. Scalable near identical image and shot detection. In *Proceedings of the 6th ACM international conference on Image and video retrieval*, pages 549–556. ACM, 2007.

[3] J. Hare, S. Samangooei, and D. Dupplaw. Openimaj and imageterrier: Java libraries and tools for scalable multimedia analysis and indexing of images. In *Proceedings of the 19th ACM international conference on Multimedia*, pages 691–694. ACM, 2011.

[4] C. Zauner. *Implementation and Benchmarking of Perceptual Image Hash Functions*. PhD thesis, MasterâĂŹs thesis, University of Applied Sciences Upper Austria, 2010.