’pdfaer’

Oliver Fong

August 22, 2025

Contents

Introduction

This simple project is meant to provide a library and executable which converts PDFs into PDF/A-compliant files for the University of Chicago’s DLDC attachment converter project. This project is essentially an extension of John Whitington’s Camlpdf library, while also meshing it with Ezxmlm, a version of Xmlm supposedly easier to understand and work with. The program runs as follows: 1) Look for non-compliancy - which parts of the PDF are non-compliant with the PDFA standards? 2) For whichever aspects of the file are non-compliant, then are they either implemented in the PDF but wrong, or are they simply absent from the file altogether? In the latter case, create the missing piece, insert the relevant objects, and insert/modify references to them from depending objects. In the former case, then parse the existing object, look for what the error is, modify it, and reinsert the modified object altogether. Do this for all pertinent problems, and return the edited file.”) (synopsis ”Simple API and executable for converting pdf files into pdfa-compliant files.

Architecture

The project is organized using several modules:

- **fonts.ml**: Logic and definitions for representing and handling fonts in digital documents.
- **lib.ml**: Provides the core library interface, connecting different parts of the system.
- **outintent.ml**: Manages output intent metadata, something that is often necessary to add to a non-compliant PDF.
- **packet.ml**: Encapsulates XMP packet structures for working with XML and RDF tags. Uses Ezxmlm to interface with existing XML, or to create and insert whatever tags or metadata necessary.
- **utility.ml**: Contains helper functions and reusable routines that support the main modules.

Implementation Approach

The design follows separation of concerns, with each module serving a clear role. I think the best way to go about organizing this project further is by keeping each specification of PDF/A separate, such that excryption and decryption is its own module and so on and so forth.

Building

The project uses Dune for compilation:

```
; Include this stanza in lib/dune for
; custom repl that works with camlpdf
(toplevel
 (name repl)
 (libraries camlpdf ezxmlm xmlm))
```

To build the project

```
$ opam exec -- dune build
```

To use with custom REPL

```
$ opam exec -- dune build lib/repl.exe
```

To use utop as the REPL instead.

```
$ opam exec -- dune ocaml top > .ocamlinit
$ utop
```

Testing

To use the python testing suite:

```
project_root/testing $ python{3} -m tests
; OR
project_root/testing $ python3 tests/__main__.py
```

I've organized the python tests into these modules:

- **exec_tools.py**: Contains functions to run **verapdf** and **pdfaer.exe**, and looks for the latter in the _build/app/ directory. As of now, the .icc file isn't being supplied to the executable as an option.
- **filter_verapdf.py**: This one supplies some simple regex functionality to filter through verapdf's XML output. Is the new library file which used to be process_verapdf.py.
- **filescript.py**: Contains helper functions for trying different common filepaths for the expected executable file and .icc file.
- **__main__.py**: The actual runfile - first converts all the pdfs in the infiles directory into pdfa files in the outfiles directory, then runs verapdf on them all compiling them in a date-timestamped logfile in logs/

Discussion

Strengths: clean modular structure, type safety, reproducible builds, semi-automated testing suite, features mulitple layers of abstraction/

Limitations: domain-specific scope, esoteric user-facing documentation, especially as it pertains to camlpdf.

Future Work: The most important features to add now are as such

- Finish implementing font embedding functionality. In this includes embedding standard Base14 fonts, CID fonts, proprietary fonts, etc. I believe this will be the toughest feature to implement because of the wide array of font types.
- Decrypting encrypted PDFs. All PDF/A-compliant PDFS must be decrypted.
- Dealing with compressed PDFs. This will likely involve just decompressing the PDF to modify it then recompressing it with a compliant codec. However, I'm not sure if recompressing will be necessary, as keeping the PDF would more or less fit better into PDF/A's ideological framework.
- Extending the testing suite to make it fully automatic, so that it is incorporated into dune's sandbox. Also, changing the logging method could be good for log legibility.

This project highlights how OCaml and Dune can be combined to create a modular, type-safe system for managing fonts, XML (XMP Metadata), and output intent metadata. It provides a foundation for further development into a more general-purpose data-conversion toolkit.

Example Code

REPL

```
(* Module: Utility *)
fname_remove_extension (s:string) : string
# fname_remove_extension "../document.pdf" :
"document"

rev_dict_entries (dict:pdfobject) : pdfobject
# rev_dict_entries Dictionary [("a", Name "b"); ("c", String "d")]
Dictionary [("c", String "d"); ("a", Name "b")]
# rev_dict_entries (Indirect 5)
Dictionary []

(* Module: *)
```