lab6

In This Issue

£1.99 JULY 2021

ISSUE 2

A Gemini-PDF polyglot

If you enjoyed last month's PDF-MP3 polyglot doc-cast, you're going to love this hybrid Gemini-PDF file. It's a single file that parses as a well-formed and valid PDF/A document with embedded images and fonts, and also as a UTF-8 text file in text/gemini format. The file depends on no external resources and operates perfectly offline. PDF/A forbids active content such as JavaScript, Flash, and forms, and has no support for any surveillance of users, while still allowing graphics and pagination.

This file, served as application/pdf over https://
This file, served as text/gemini over gemini://

Why?!

Gemini achieves its power-to-weight ratio through the simplicity of text, foregoing other media types. But if text is the system of record of our current civilisation, then imagery is the system of engagement. Sometimes a thousand words will do, and sometimes you need an image. But text is just glyphs, and glyphs are just images with pre-agreed meaning. And so text possesses no inherent simplicity advantage over images, despite how our UNIXish culture makes text seem like the simpler option. Text and images are both ultimately just pixels drawn in grids. In fact, text layout and font rendering is considerably more complex than painting a raster image to the screen. Gemini pushes the complexity of text rendering into the client, which is the same tradeoff that HTML makes. This pattern, of separating content-as-data from presentation-as-application, is deeply ingrained across nearly all of the computing landscape. It does make a lot of sense in most contexts: applications come and go, but data lives forever.

But on the subject of forever... when it comes to the archival and preservation context, it is no longer enough to have just the data. In general, it is risky to rely on today's applications being available in the future. Hardware evolves, operating systems evolve, APIs and ABIs and dependencies all evolve, and commitments to backwards compatibility weaken, and are eventually forgotten. A lot of this is malign churn, but change is inevitable.

Should we just archive our applications alongside our data, along with an emulation environment? Yes, we should definitely do that. But it's not the only approach. PDF/A makes a different design tradeoff to the content-as-data file formats, by being highly opinionated on how the content should be presented. Of course a PDF viewing application is still necessary, but the PDF spec gives a much stronger guarantee that such an application will render the content consistently across platforms, and across time. Archivists don't trust that future software will be able to read today's data properly, making PDF/A an attractive proposition for the long-term preservation of content.

On the other hand, maybe content authors don't care how their work is consumed, or prefer users to have a choice, or hope that future software will be better than today's software and presentation will change over time for the better. These are also valid perspectives.

Could we just publish content in multiple parts and multiple formats? A simple text-based format for dynamic presentation, with some images alongside, and another file combining them into a static fixed-layout format? Surely yes, but we then must give up an important benefit: the encapsulation of content into a singular file with unambiguous identity. Perhaps you are not sold on this benefit? It is a fantastically useful feature for users, and even enables a fundamental freedom: the freedom to possess the content. It very strongly guarantees that the parts are not separated from the whole. Who has not come across a copy of an HTML file with dead or broken links to inline images? Encapsulation is why we distribute software as containers, and why we embed metadata into JPEGs, and why we multiplex video and audio into media container formats. Lack of encapsulation is why XMP sidecar files are fragile, and why DLL hell happens, and why URLs rot.

This principle of self-containment is what gives you immutability, archiveability, and offlineability. A document is not a complete document if important constituent parts of it are stored elsewhere. Links eventually break, and loosely-coupled parts may diverge over time. Furthermore, fragmenting content into linked chunks is an opportunity for user abuse: "Click here for page 2! (and don't mind the interstitial ads)".

You've only got it if you can hash it.

What about just shipping content in multiple formats and wrapping all the parts up in a zip file? Sure, that could work, except that would effectively be defining a new format which would require viewer applications to change, or manual intervention from the user to unpack first. Just take a look at the history of trying to get users to adopt new formats or new anything. Any barrier can be fatal.

Since we like Gemini so much, could we just extend Gemini to include images, if that's what we really care about? Maybe inline base64-encoded? Cue gnashing of teeth from those who know better than to reopen a Door That Has Been Closed. Gemini is finished, and by design it should not be extended.

So let's recap some requirements:

- Dynamic presentation of content according to the user's wishes
- Static presentation of content according to the author's wishes
- Self-contained file format
- Opens with zero friction in existing document viewing applications
- Fully specified
- A safe space away from malware and dark patterns

And so we arrive at the answer: you're looking at it!

PDF/A shares many goals with Gemini: no dynamic content, a fixed scope, an ecosystem of compatible tools, and it's primarily and fundamentally a document format, not a web application. Both have a detailed spec and an institutional commitment to stability (insofar as Gemini has any institutions). Although the typical deployment of PDF/A is long-form texts - articles, papers, books, reports - the format itself is not exactly a text format, but rather a language for painting pixels. It has text drawing and transformation and graphics operators, derived from PostScript, and it manifests as a binary file format with internal pointers to exact offsets. It's vastly more complex than Gemini, but its power-to-weight-ratio impresses given the vastly increased scope. A lot of what goes on in a PDF file is a reflection of the complexity of document rendering that in HTML would be delegated to the browser. The PDF 1.7 spec is 756 pages long, but that's actually astonishingly concise and restrained compared to the fractal sprawl of web tech specs.

Read more about why the static, self-contained nature of PDFs is such a powerful antidote for web-itis.

Why PDF/A and not just plain PDF? Because the /A profile willingly forgoes features that are difficult or impossible to archive, and unwittingly strips out all of the bloat, churn, and danger that pollutes the the HTML-based web. No JavaScript, no forms, no Flash, no video, no audio, no actions, no executable code, and most importantly no reliance on outside resources. PDF/A should allow PDF viewers to enter a more restricted processing mode with a greatly reduced attack surface. PDF/A also mandates use of tags which assist assistive tools like screen readers. And miraculously, PDF/A already exists today! It's done - no need for new work on new specs. It's all ISO-standardised and interoperable and just there ready for the taking. Naturally, it's imperfect and has enterprisey smells, but who doesn't?

PDF/A is not presented here as an extension of Gemini - it's the enriched prequel.

In short: Gemini & PDF/A, together at last.

How?

If PDFs are binary objects with pointers and compressed streams, how can this document possibly be a a valid UTF-8 Gemini text file?

There are many mini-tutorials available which show how to create PDF files from scratch, using plain text files, such as the following:

Inside the PDF File Format
Understanding the Portable Document Format
Ange Albertini's visual PDF format walkthrough

They tend to bail out before getting to PDF/A, because PDF/A immediately brings in a whole bunch of complication. As a PDF/A is not allowed to depend on external resources, you have to embed everything:

- You need to embed every font you use, but you also probably want to subset each of those fonts to only include the glyphs that your document uses, in order to reduce file size (Unicode fonts can be large).
- As well as the actual font program, you need to supply quite a lot of font metadata so that viewers can efficiently paint glyphs and extract text.
- You also need to satisfy the tagging requirements, which means building a parallel structure tree containing or referencing all content.
- You even need to embed a colour profile, so that compliant viewers can exactly reproduce the content on a future display device.

The main trick of producing this polyglot file is that PDF doesn't care at all what data is in the file as long as viewers can find specific objects at specific offsets. Any old data (such as a Gemini-formatted stream of text) can go in between the objects.

We start the file with the following mandatory PDF magic file bytes:

```
%PDF-1.7
```

The next line starts with the PDF comment character (%) and is required to contain at least 4 consecutive bytes with the high-order bit set, in order to signal that the file is not 7-bit ASCII. In this file we use U+264A and U+1F5CE - the symbols for "gemini" and "document" respectively. These two lines are ordinary text lines when interpreted as Gemini. What follows is the Gemini content. We could attempt to comment this out using % characters, but there is no point, because PDF processors read PDF files backwards: they start at the end, where a lookup table is held. That lookup table (xref table) gives pointers to the start of each object. We simply don't have any objects pointing to the Gemini content, so it gets seek()'d past.

When we finish the Gemini content, we need to switch on preformatting mode with three backticks ``` so that the whole set of PDF objects is rendered by Gemini clients as a block of preformatted text, as though it were ASCII art. This is something of a downside, as Gemini clients have no use for any of this data, which may as well be trailing garbage. Authors will simply have to state that the meaningful content of the Gemini document has finished and what follows can be safely ignored.

We have to end the Gemini preformatted block early, because PDF processors expect the end of the file to contain a master pointer to the main object offset lookup table, like this:

```
startxref
314521
%%EOF
```

These will be rendered as plain Gemini text lines, but users should have given up trying to read this far down anyway.

What about binary blobs like images and fonts? Luckily, PDF supports the concept of filters. Each object can be optionally encoded by a chain of filters. One of those filters is Flate, for compressing data. Another is ASCII85, for encoding data as ASCII characters. By using these two filters together, in that order, we can save some bytes through compression, and then re-waste some bytes expanding them out to ASCII again.

There's one final piece of the puzzle: since the backtick ` is a character in the ASCII85 set, might we end up with a line beginning with three of them, thereby accidentally toggling the Gemini client's preformatting mode off? Yes, we might: the byte sequence 0xC6 0x5A will encode to ```. We can't rule out the possibility of that appearing in a binary stream. Fortunately, PDF is forgiving of whitespace, and so all we have to do is break up any occurrence of ``` by inserting a space in amongst the backticks.

None of this is intended to be exploitative of Gemini or PDF clients. The file validates flawlessly in verapdf as conformance level PDF/A-3b, even though it takes a daring shortcut when generating the structure tree (all content is given as /ActualText entries rather than attempting to mark up the actual in-stream content using /MCID tags). Consequently, it doesn't validate as PDF/UA, but perhaps the plain text Gemini structure makes up for that.

Look on the coverdisk (a PDF attachment) for the thousand nauseating lines of spaghetti Python that bent time and space to generate this file. You won't find a finer or fouler PDF/A-from-scratch POC. Enjoy!

P.S. The cover background is a small slice into the mandelbrot set, rendered using PDF graphics operators, not an image. Zoom in for vector smoothness!

Some gory facts about PDF/A internals

PDFs are easiest to generate in-memory before serializing, as there are many pairs of objects that refer to each other by ID.

PDF metadata has serious second-system syndrome. There is a legacy object type for storing basic metadata like document title and document author. Clearly somebody hit the limitations of these simple metadata fields, and went full RDF on its successor. You never go full RDF.

PDFs automate almost nothing about the layout and presentation. Fonts have no intrinsic concept of underlining, so hyperlinks need to be printed, and then a blue line drawn under them separately. Every piece of text has to be placed at (x,y) coordinates of your choosing. There is no automatic line wrapping or word breaking or justification. As a result, the typesetting of this document probably looks better in Gemini mode in Lagrange than in a PDF viewer.

The Tilde Quilt

Not content with launching just one brand new media paradigm per issue, Lab 6 additionally brings you The Tilde Quilt:

2-dimensional anonymous collaborative microblogging

This combines the exclusivity of private pubnix membership, the value-hoarding artificial scarcity of sequential integers, the inspiration of spatial constraint, the north star of the unbounded cartesian grid, and the immediacy of named pipes.

It looks a little like this:

```
+------------------------------++------------------------------+
|26            2021-03-30 15:03||18            2021-02-03 20:25|
|Wake up before noon           ||,            _____            |
|                              ||||'.      ~~/_0_0_\         .,|
|                              ||#**;',.      '   '      .,;*|#|
|                              ||#######;:,'.,;':..;:;,:;#|#|##|
|                              ||##### I WANT TO BELIEVE ######|
+------------------------------++------------------------------+
+------------------------------++------------------------------+
|9             2021-01-31 09:53||10            2021-01-31 09:58|
|>>> Head east to play with >>>||          _       _      __   |
| >>> ASCII art, creative    >>||    _    | | __ _| |__  / /_  |
|> >>> writing, poetry, and   >||  _| |_  | |/ _` | '_ \| '_ \ |
|>> >>> that sort of thing... || |_   _| | | (_| | |_) | (_) ||
|>>> >>>>>>>>>>>>>>>>>>>>>>>>>>||   |_|   |_|\__,_|_.__/ \___/ |
+------------------------------++------------------------------+
```

Grab an archived snapshot of the current state of the quilt on the coverdisk, or visit the link above to see it live.

Society & co

Relevant

[Terminaltables - generate ASCII tables using Python]
[Lagrange: A Beautiful Gemini Client]
[JHOVE - PDF validation]
[veraPDF - PDF/A validation]
[PAC3 PDF Accessibility Checker]
[moreutils, including the isutf8 command]

Contemporary

[Paged Out zine]
[ASCII Town]
[1MB club - some issues of Lab 6 may qualify.]
[Bryan Ropar's Plastic Chair World]
[Reverse Engineering the iOS Backup]
[NNCP: Lossless Data Compression with Neural Networks]
[SQLite over HTTP - another static success story]
[Null Programs and the Uninscribed]

Classics

[Starship Dimensions]
[Syndicate Wars Port]
[British Walks, a one-man op since 1999]
[Suicide Linux]

Comments

Send your feedback to comment@input.lab6.com and it may be published in the next issue.

You can also comment on the issue by typing your comment into the URL bar of your browser, immediately after the issue number, like this:

```
https://www.lab6.com/2Your Comment Goes Here
gemini://lab6.com/2or here.
```

(read more on page 7 of the last issue for how this works)

Only one reader has figured out how to use this mechanism so far, leaving this comment on Issue 1:

> Windows says coverdisk is corrupted for both issues

Sadly, as of version 10, Windows no longer knows how to mount FAT12-formatted floppy disk images. Linux will do it happily though:

```
$ sudo mount coverdisk0.img /mnt
```

Or you can use GNU mtools to extract files without mounting:

```
$ mdir -i coverdisk0.img
Volume in drive : has no label
Volume Serial Number is AAB9-CC36
Directory for ::/

PUBLIC   KEY      3130 2021-01-02  23:18
FORENSIC ZIP     21162 2021-01-02  23:18
2 files               24 292 bytes
1 432 576 bytes free

$ mcopy -i coverdisk0.img ::/FORENSIC.ZIP .
```

GNU mtools

Back Issues

Issue 0

All hail PDF; FORENSIC.ZIP.

Lab 6 Issue 0 over HTTPS
Lab 6 Issue 0 in IPFS
Lab 6 Issue 0 in Geminispace (in PDF format)

```
SHA-256: 00c411fee9419cd861d9850dc56d53b7e6a211e90df9a1ca953e021b0cf31a56
```

Issue 1

Making PDFs accessible by polyglotting with MP3; Remarks on replacing HTML with PDF; Fantasy Footbyte, with 256 unique team name puns.

Lab 6 Issue 1 over HTTPS
Lab 6 Issue 1 in IPFS
Lab 6 Issue 1 in Geminispace (in PDF/MP3 format)

```
SHA-256: 0145df4ffcd382db238d1bc87c014013aa9cbc2298fe2a68fef09ca66cc99da6
```

```
ø¤º°`°º¤ø,¸,ø¤º°`°º¤ø,¸,ø¤º°`°º¤ø,¸,ø¤º°`°º¤ø,¸,ø¤º°`°º¤ø,¸,ø¤º°`°º¤ø,¸,
```

This document's hexadecimal SHA-256 hash begins 0x02, identifying the issue number.

Lab 6 is the official sponsor of 6

Tracking

OEIS entries	346,036
English Wikipedia entries	6,337,397
RC5-72 keys tested	370,643,516,324,642,291,712
Bitcoin blocks	691,133
XKCDs	2,489
IPv6 adoption	36.53%
Abe Vigoda Status Page Status	Still up
Largest Prime	$2^{82589933} - 1$
BTTF Movies	3
World population according to the CIA	7,772,850,805
Latest stable Linux kernel	5.13.2
Color Names	2,586,343
Latest number of tildeverse's #counting	8,384
Count of University of Queensland pitch drops	9
Project Gutenberg eBooks	65,820
OCRemix Releases	4,050

If you're reading this in a PDF viewer, congratulations, you made it to the end!
If you're reading this in a Gemini client, what follows is the PDF data.