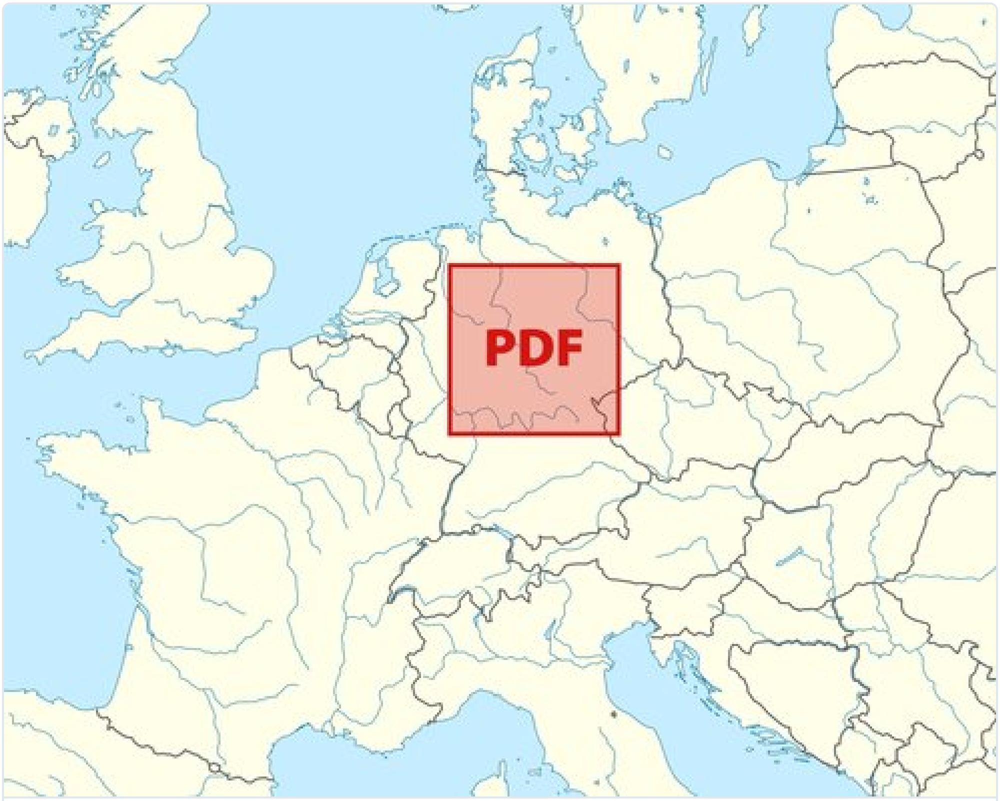
Making a PDF that’s larger than Germany

Posted 31 January 2024 · Tagged with code-crimes, drawing-things

I was browsing social media this morning, and I saw a claim I’ve seen go past a few times now – that there’s a maximum size for a PDF document:



Terrible Maps
@TerribleMaps

Maximum size of a PDF, version 7: 381 km × 381 km.

https://commons.m.wikimedia.org/wiki/File:Seit…
5:14 PM - 30 Jun 2023

Some version of this has been floating around the Internet since 2007, probably earlier. This tweet is pretty emblematic of posts about this claim: it’s stated as pure fact, with no supporting evidence or explanation. We’re meant to just accept that a single PDF can only cover about half the area of Germany, and we’re not given any reason why 381 kilometres is the magic limit.

I started wondering: has anybody made a PDF this big? How hard would it be? Can you make a PDF that's even bigger?

A few years ago I did some silly noodling into PostScript, the precursor to PDF, and it was a lot of fun. I've never actually dived into the internals of PDF, and this seems like a good opportunity.

Let's dig in.

Where does the claim come from?

These posts are often accompanied by a "well, actually" where people in the replies explain this is a limitation of a particular PDF reader app, not a limitation of PDF itself. They usually link to something like the Wikipedia article for PDF, which explains:

> Page dimensions are not limited by the format itself. However, Adobe Acrobat imposes a limit of 15 million by 15 million inches, or 225 trillion in^2 (145,161 km^2).[2]

If you follow the reference link, you find the specification for PDF 1.7, where an appendix item explains in more detail (emphasis mine):

> In PDF versions earlier than PDF 1.6, the size of the default user space unit is fixed at 1/72 inch. In Acrobat viewers earlier than version 4.0, the minimum allowed page size is 72 by 72 units in default user space (1 by 1 inch); the maximum is 3240 by 3240 units (45 by 45 inches). In Acrobat versions 5.0 and later, the minimum allowed page size is 3 by 3 units (approximately 0.04 by 0.04 inch); the maximum is 14,400 by 14,400 units (200 by 200 inches).
>
> Beginning with PDF 1.6, the size of the default user space unit may be set with the UserUnit entry of the page dictionary. **Acrobat 7.0 supports a maximum UserUnit value of 75,000, which gives a maximum page dimension of 15,000,000 inches (14,400 * 75,000 * 1⁄72).** The minimum UserUnit value is 1.0 (the default).

15 million inches is exactly 381 kilometres, matching the number in the original tweet. And although this limit first appeared in PDF 1.6, it's "version 7" of Adobe Acrobat. This is probably where the original claim comes from.

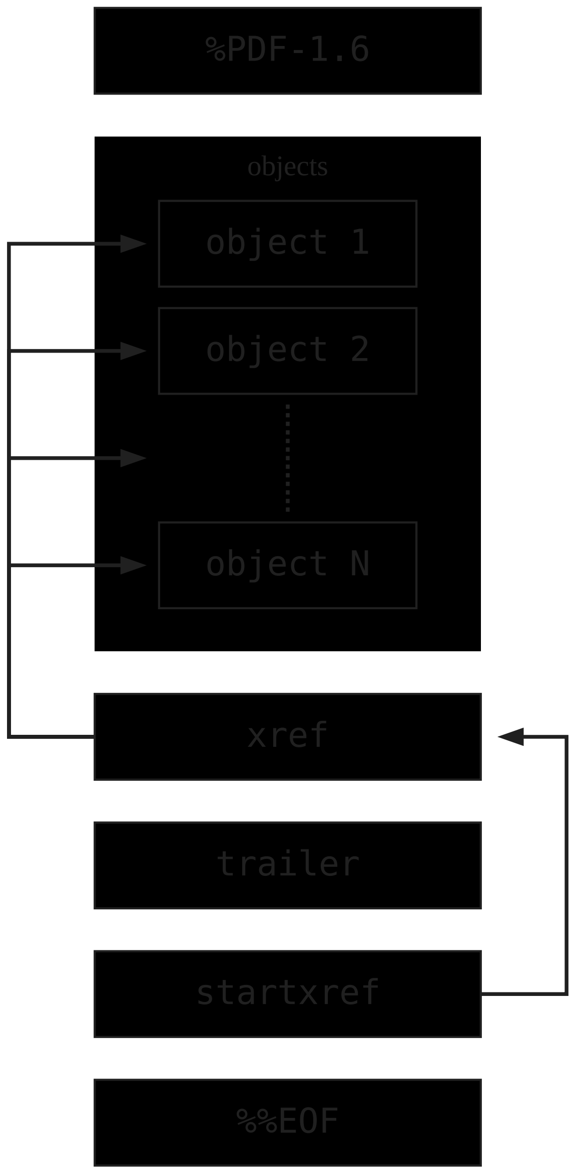
What if we make a PDF that exceeds these "maximum" values?

The inner structure of PDFs

I’ve never dived into the internals of a PDF document – I’ve occasionally glimpsed some bits in a hex editor, but I’ve never really understood how they work. If I’m going to be futzing around for fun, this is a good opportunity to learn how to edit the PDF directly, rather than going through a library.

I found a good article which explains the internal structure of a PDF, and combined with asking ChatGPT a few questions, I was able to get enough to write some simple files by hand.

I know that PDFs support a huge number of features, so this is probably a gross oversimplification, but this is the mental picture I created:



The start and end of a PDF file are always the same: a version number (`%PDF-1.6`) and an end-of-file marker (`%%EOF`).

After the version number comes a long list of objects. There are lots of types of objects, for all the various things you can find in a PDF, including the pages, the text, and the graphics.

After that list comes the `xref` or cross-reference table, which is a lookup table for the objects. It points to all the objects in the file: it tells you that object 1 is 10 bytes after the start, object 2 is after 20 bytes, object 3 is after 30 bytes, and so on. By looking at this table, a PDF reading app knows how many objects there are in the file, and where to find them.

The `trailer` contains some metadata about the overall document, like the number of pages and whether it's encrypted.

Finally, the `startxref` value is a pointer to the start of the `xref` table. This is where a PDF reading app starts: it works from the end of the file until it finds the `startxref` value, then it can go and read the `xref` table and learn about all the objects.

With this knowledge, I was able to write my first PDF by hand. If you save this code into a file named `myexample.pdf`, it should open and show a page with a red square in a PDF reading app:

```
%PDF-1.6

% The first object.  The start of every object is marked by:
%
%     <object number> <generation number> obj
%
% (The generation number is used for versioning, and is usually 0.)
%
% This is object 1, so it starts as `1 0 obj`.  The second object will
% start with `2 0 obj`, then `3 0 obj`, and so on.  The end of each object
% is marked by `endobj`.
%
% This is a "stream" object that draws a shape.  First I specify the
% length of the stream (54 bytes).  Then I select a colour as an
% RGB value (`1 0 0 RG` = red), then I set a line width (`5 w`) and
% finally I give it a series of coordinates for drawing the square:
%
%     (100, 100) ----> (200, 100)
%                           |
%     [s = start]           |
%         ^                 |
%         |                 |
%         |                 v
%     (100, 200) <---- (200, 200)
%
1 0 obj
```

```
<<
        /Length 54
>>
stream
1 0 0 RG
5 w
100 100 m
200 100 l
200 200 l
100 200 l
s
endstream
endobj

% The second object.
%
% This is a "Page" object that defines a single page.  It contains a
% single object: object 1, the red square.  This is the line `1 0 R`.
%
% The "R" means "Reference", and `1 0 R` is saying "look at object number 1
% with generation number 0" -- and object 1 is the red square.
%
% It also points to a "Pages" object that contains the information about
% all the pages in the PDF -- this is the reference `3 0 R`.
2 0 obj
<<
        /Type /Page
        /Parent 3 0 R
        /MediaBox [0 0 300 300]
        /Contents 1 0 R
>>
endobj

% The third object.
%
% This is a "Pages" object that contains information about the different
% pages.  The `2 0 R` is reference to the "Page" object, defined above.
3 0 obj
<<
        /Type /Pages
        /Kids [2 0 R ]
        /Count 1
>>
endobj

% The fourth object.
%
% This is a "Catalog" object that provides the main structure of the PDF.
% It points to a "Pages" object that contains information about the
% different pages -- this is the reference `3 0 R`.
4 0 obj
<<
        /Type /Catalog
        /Pages 3 0 R
>>
endobj

% The xref table.  This is a lookup table for all the objects.
%
% I'm not entirely sure what the first entry is for, but it seems to be
% important.  The remaining entries correspond to the objects I created.
xref
0 4
0000000000 65535 f
0000000851 00000 n
0000001396 00000 n
0000001655 00000 n
0000001934 00000 n

% The trailer.  This contains some metadata about the PDF.  Here there
% are two entries, which tell us that:
%
%   - There are 4 entries in the `xref` table.
%   - The root of the document is object 4 (the "Catalog" object)
%
trailer
<<
        /Size 4
        /Root 4 0 R
>>

% The startxref marker tells us that we can find the xref table 2196 bytes
% after the start of the file.
startxref
2196

% The end-of-file marker.
%%EOF
```

I played with this file for a while, just doing simple things like adding extra shapes, changing how the shapes appeared, and putting different shapes on different pages. I tried for a while to get text working, but that was a bit beyond me.

It quickly became apparent why nobody writes PDFs by hand – it got very fiddly to redo all the lookup tables! But I'm glad I did it; manipulating all the PDF objects and their references really helped me feel like I understand the basic model of PDFs. I opened some "real" PDFs created by other apps, and they have many more objects and types of object – but now I could at least follow some of what's going on.

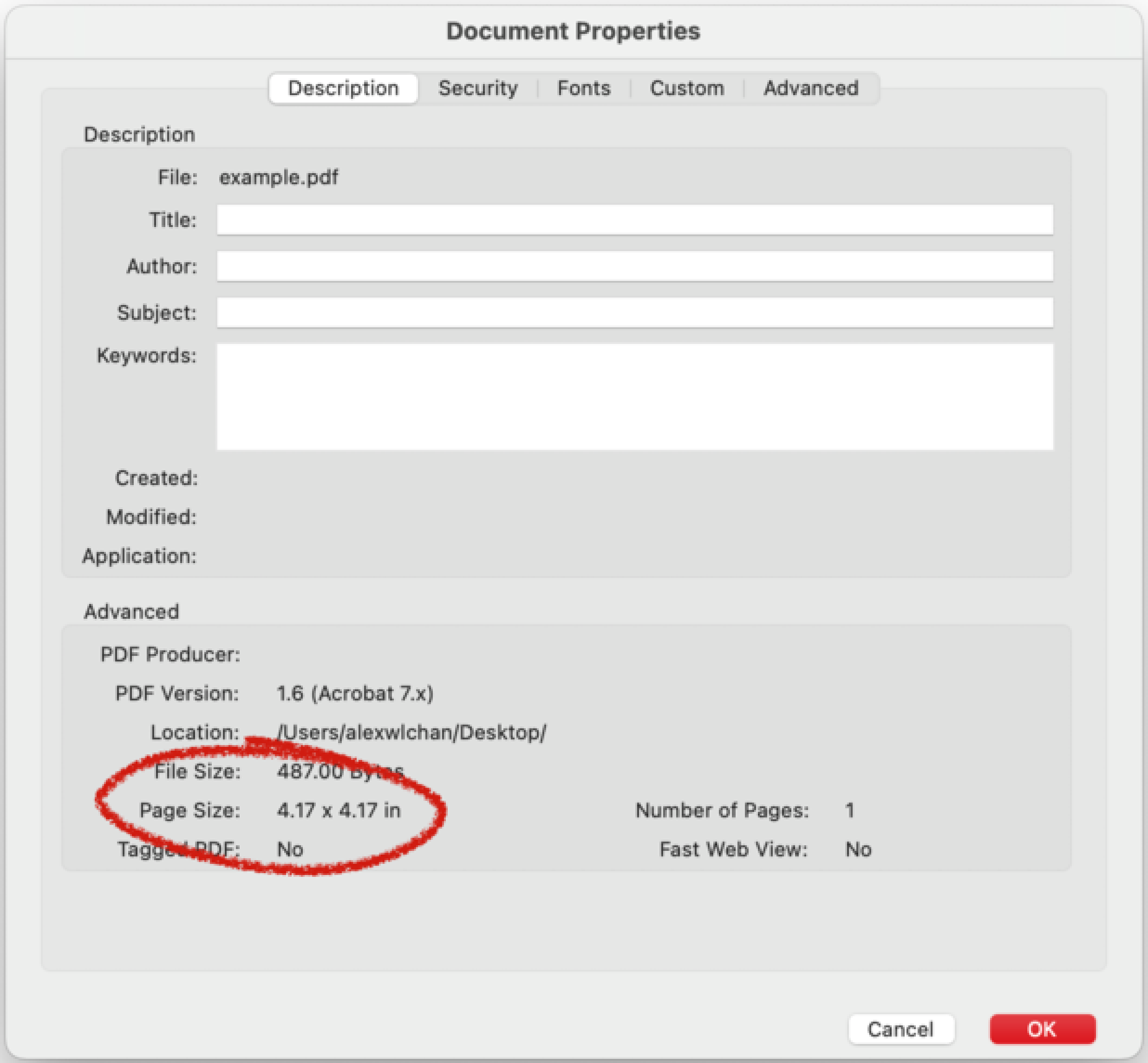
With this newfound ability to edit PDFs by hand, how can I create monstrously big ones?

Changing the page size: /MediaBox and /UserUnit

Within a PDF, the size of each page is set on the individual "Page" objects – this allows different pages to be different sizes. We've already seen this once:

```
<<
        /Type /Page
        /Parent 3 0 R
        /MediaBox [0 0 300 300]
        /Contents 1 0 R
>>
```

Here, the `MediaBox` is setting the width and height of the page – in this case, a square of 300 × 300 units. The default unit size is 1/72 inch, so the page is 300 × 72 = 4.17 inches. And indeed, if I open this PDF in Adobe Acrobat, that's what it reports:


Document Properties

Description

File: example.pdf
Title:
Author:
Subject:
Keywords:

Created:
Modified:
Application:

Advanced

PDF Producer:
PDF Version: 1.6 (Acrobat 7.x)
Location: /Users/alexwlchan/Desktop/
File Size: 487.00 Bytes
Page Size: 4.17 x 4.17 in
Number of Pages: 1
Tagged PDF: No
Fast Web View: No

By changing the `MediaBox` value, we can make the page bigger. For example, if we change the value to `600 600`, Acrobat says it's now `8.33 x 8.33 in`. Nice!

We can increase it all the way to `14400 14400`, the max allowed by Acrobat, and then it says the page is now `200.00 x 200.00in`. (You get a warning if you try to push past that limit.)

But 200 inches is far short of 381 kilometres – and that's because we're using the default unit of 1/72 inch. We can increase the unit size by adding a `/UserUnit` value. For exaple, setting the value to 2 will double the page in both dimensions:

```
<<
        /Type /Page
        /Parent 3 0 R
        /MediaBox [0 0 14400 14400]
        /UserUnit 2
        /Contents 1 0 R
>>
```

And now Acrobat reports the size of the page as `400.00 x 400.00 in`.

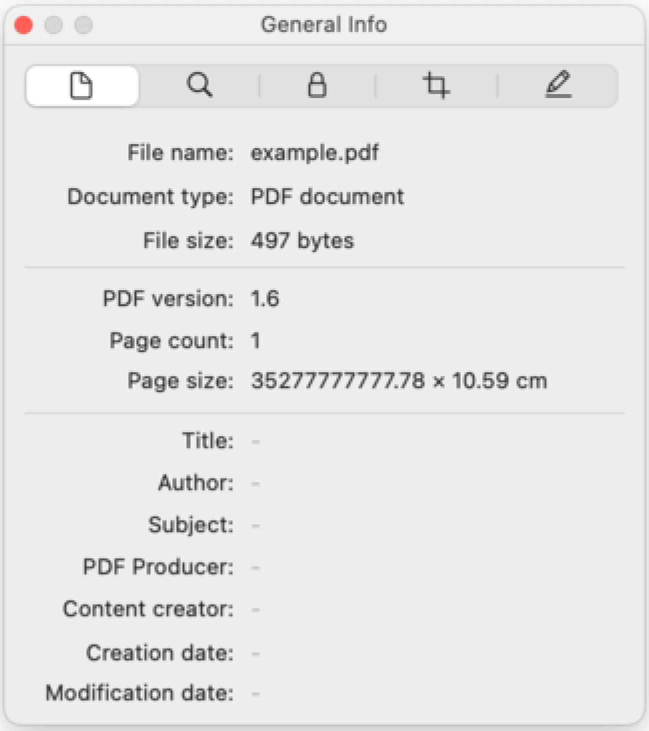
If we crank it all the way up to the maximum of `UserUnit 75000`, Acrobat now reports the size of our page as `15,000,000.00 x 15,000,000.00 in` – 381 km along both sides, matching the original claim. If you're curious, you can download the PDF.

If you try to create a page with a larger size, either by increasing the `MediaBox` or `UserUnit` values, Acrobat just ignores it. It keeps saying that the size of a page is 15 million inches, even if the page metadata says it's higher. (And if you increase the `UserUnit` past `75000`, this happens silently – there's no warning or error to suggest the size of the page is being capped.)

[**Edit, 1 February 2024:** some extra zeroes slipped into the original version of this post – it's a million inches, not a billion. Thanks to mrb on Hacker News for spotting the mistake!]

This probably isn't an issue – I don't think the `UserUnit` value is widely used in practice. I found one Stack Overflow answer saying as such, and I couldn't find any examples of it online. The builtin macOS Preview.app doesn't even support it – it completely ignores the value, and treats all PDFs as if the unit size is 1/72 inch.

But unlike Acrobat, the Preview app doesn't have an upper limit on what we can put in `MediaBox`. It's perfectly happy for me to write a width which is a 1 followed by twelve 0s:



File name: example.pdf
Document type: PDF document
File size: 497 bytes
PDF version: 1.6
Page count: 1
Page size: 35277777777.78 × 10.59 cm
Title: -
Author: -
Subject: -
PDF Producer: -
Content creator: -
Creation date: -
Modification date: -

If you're curious, that width is approximately the distance between the Earth and the Moon. I'd have to get my ruler to check, but I'm pretty sure that's larger than Germany.

I could keep going. And I did. Eventually I ended up with a PDF that Preview claimed is larger than the entire universe – approximately 37 trillion light years square. Admittedly it's mostly empty space, but so is the universe. If you'd like to play with that PDF, you can get it here.

Please don't try to print it.