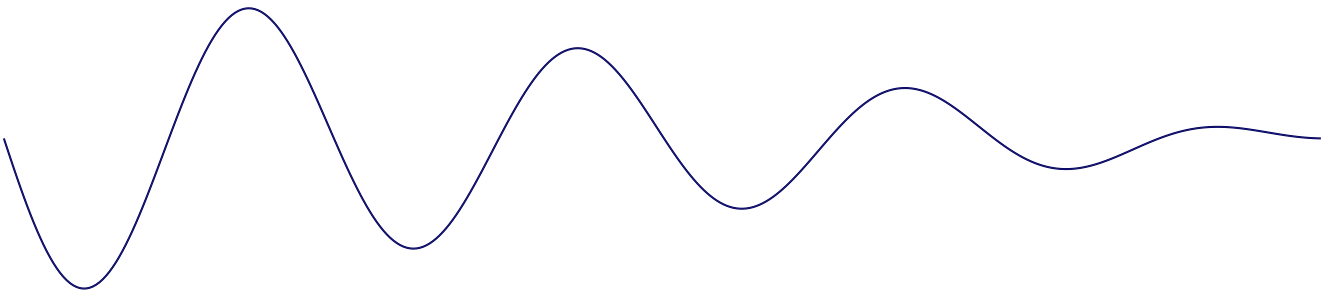
Representative content

Compression acts on the content stream, which is the list of drawing operators a page is made of. Text is cheap, paths are dear, and an image is usually already compressed by the time it arrives - which is why the settings below move the total by wildly different amounts.

Compression acts on the content stream, which is the list of drawing operators a page is made of. Text is cheap, paths are dear, and an image is usually already compressed by the time it arrives - which is why the settings below move the total by wildly different amounts.

Compression acts on the content stream, which is the list of drawing operators a page is made of. Text is cheap, paths are dear, and an image is usually already compressed by the time it arrives - which is why the settings below move the total by wildly different amounts.

Compression acts on the content stream, which is the list of drawing operators a page is made of. Text is cheap, paths are dear, and an image is usually already compressed by the time it arrives - which is why the settings below move the total by wildly different amounts.



What each setting costs

The page before this one, saved eight times under different options. Every one of those files renders identically; the only difference between them is how many bytes it takes to say the same thing. That is why this demo is a table - there is nothing else to look at.

setting	bytes	against the first row
`CompressContentStreams = true` The baseline every other row is measured against	289,120	-
`CompressContentStreams = false` Operators written as readable text	298,510	+9,390
`NoCompression = true` Nothing in the file is compressed at all	326,749	+37,629
`FlateEncodeMode.BestCompression` Slower to save, smaller to keep	289,111	-9
`FlateEncodeMode.BestSpeed` Faster to save, larger to keep	290,755	+1,635
`UseFlateDecoderForJpegImages.Always` Flate over the already-compressed JPEG, whatever it costs	289,178	+58
`UseFlateDecoderForJpegImages.Automatic` The same, kept only if it turned out smaller	289,120	-
`ColorMode.Cmyk` Four components per colour instead of three	289,119	-1
`CrossReferenceFormat.Stream` Barely moves a page that is mostly one content stream	287,638	-1,482

The default used not to be a constant

CompressContentStreams defaults to true, and now does so in every build. It used to be declared false under #if DEBUG and true otherwise, which made the size of the output a property of how the library had been compiled rather than of the code calling it: the same program wrote a materially larger PDF from a debug build, and two files could not be compared without knowing which configuration each came from. Set it to false to get the readable content stream back - that was the only thing the conditional bought.

What to make of the rest

Turning compression off has a large and predictable cost, and it is worth doing exactly once: an uncompressed PDF can be opened in a text editor and read, which is the fastest way to find out what the library actually wrote. Ship the compressed one.

UseFlateDecoderForJpegImages runs flate over a stream that is already compressed, and whether that wins depends entirely on the picture - the number above is this photograph's answer and not a general one. Always takes the flated form whatever it costs, because some readers will not decode a bare DCT stream; Automatic keeps it only when it came out smaller, which is the setting to reach for if size is the reason you are here.

ColorMode decides what is written for every colour in the file - three components or four. Switching to CMYK is what a press wants and what a screen does not; the colours are converted on the way out, so a document built in RGB and saved as CMYK is not the same colours, only the nearest ones.

One of those rows is measured unfairly

CrossReferenceFormat, which barely moves the table above and moves a great deal on the right document. That comparison is the next page, because it needs a second document to make it against.

Where a cross-reference stream pays

CrossReferenceFormat.Stream gathers the objects that may be gathered into object streams and compresses them together, and it has almost nothing to work on in the measurement on the page before: one page of drawing is a single large content stream and a dozen objects, and a content stream is not something an object stream may hold. Measured over a document that is mostly objects instead - a hundred nearly empty pages - the same setting reads differently.

format	bytes	against classic
`Classic, 100 pages`	51,320	-
`Stream, 100 pages`	27,634	-23,686
`The same page, measured twice`	289,120 / 287,638	1,482

What it costs

A reader that understands PDF 1.5, which by now is all of them - and it is the one option here PDF/A-1 refuses outright, because a cross-reference stream is a PDF 1.5 construction and PDF/A-1 is defined against 1.4. The Archive demo shows that refusal in the writer's own words.

MaxObjectsPerObjectStream trades the other way

A reader has to decompress a whole object stream to reach any one object in it, so a larger number is a smaller file and a slower open. Acrobat uses 200 and so does this. It means nothing at all unless CrossReferenceFormat is Stream, which is the sort of option that is easy to set and hard to notice has done nothing.