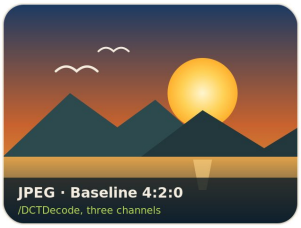
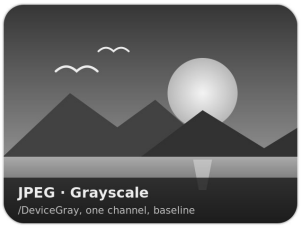
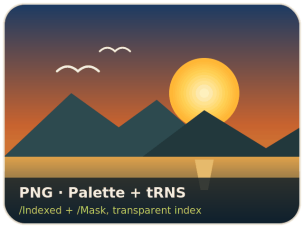
Images, gradients and patterns



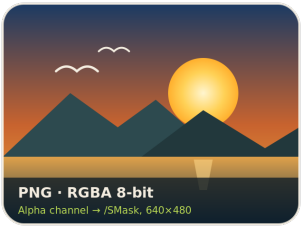
JPEG, DCTDecode
3 ms



JPEG, DeviceGray
0 ms



PNG, Indexed + Mask
1 ms



PNG, SMask
348 ms

Reuse and rotation

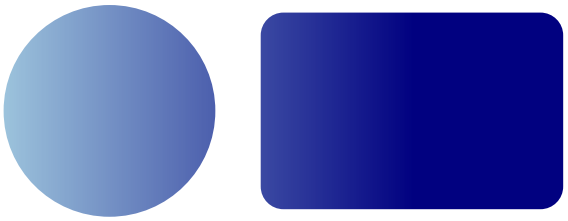
embedded once, placed seven times - by width, turned, faded, by height, squeezed by -size, and at 1200 dpi with and without -scale 0.7

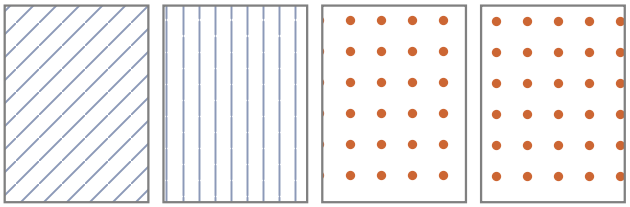
Gradients: shading types 2 and 3

axial, two colours

axial, three colours (stitched)

radial



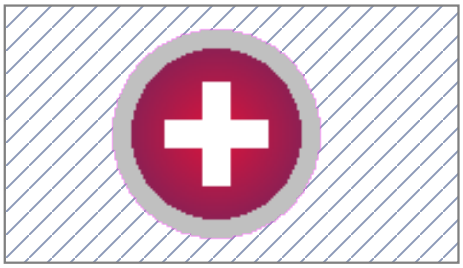


hatch, 3 mm -matrix, turned 45 dots, 4 mm -origin at the corner

tiling patterns; the tiles hang off the page - the third is cut at its left edge, the fourth begins flush at the corner it was anchored to

Colour key: one transparent colour, no alpha channel

8 bit: DeviceRGB + Mask, the ground shows through where the file said magenta (0 ms, a pass-through)

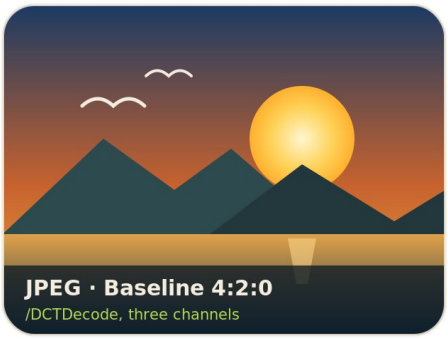


16 bit: DeviceRGB at 16 bits + SMask - a 16-bit /Mask is ignored by readers, so the key becomes a mask (3 ms, decoded once)

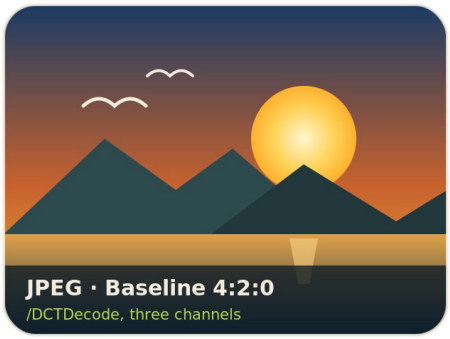
the 8-bit file on white

Pictures that arrive as bytes

A web backend never sees a file: a canvas posted from a browser, a chart from a subprocess, a BLOB out of a database. -data takes the bytes themselves. Below, the same JPEG is embedded twice - once by name, once as bytes - and the two describe themselves identically.



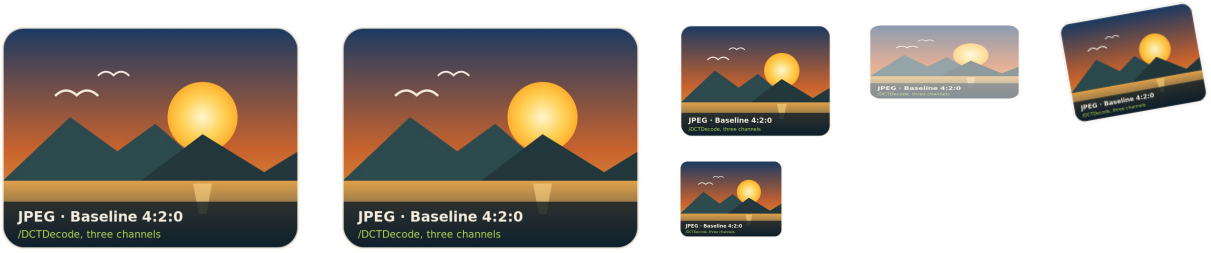
embedded by file name



embedded with -data

what the file says	by name	with -data
type	jpeg	jpeg
width	640	640
height	480	480
components	3	3
bitDepth	8	8
alpha	0	0
path	sample-photo.jpg	(empty)

The bytes are stored once even when they arrive twice: with no file name to key the cache on, the bytes themselves are the key. Two draws of the same picture below share one image object.



9 images embedded on this page and the last

Which format to ask the browser for

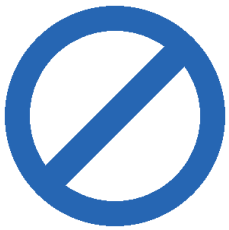
canvas.toDataURL() gives PNG with an alpha channel, which is the only path that has to be computed rather than passed through. For a chart on a white ground the channel carries nothing, and asking for image/jpeg instead turns the most expensive way in into the cheapest.

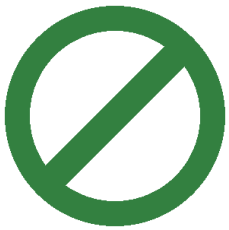
The picture that has no object

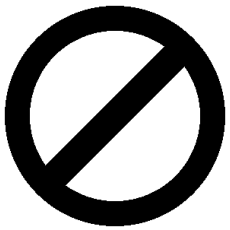
An inline image is written into the content stream: BI, an abbreviated dictionary, ID, the bytes, EI. No object, no resource entry, no reuse - and at most 4096 bytes of image data (ISO 32000-2, 8.9.7). The abbreviations are the whole of it: /W not /Width, /BPC not /BitsPerComponent, /Fl not /FlateDecode, /G not /DeviceGray.

A stencil, three times, in three colours

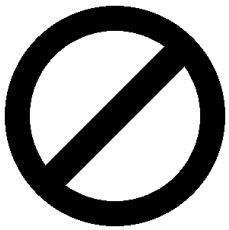
three inline stencils - three copies of the bytes







inline, /CS /G



an XObject

What the content stream holds

```
BI
/W 300 /H 300 /IM true /BPC 1 /D [0 1] /F /Fl /DP << /Predictor 15 /Colors 1 /BitsPerComponent 1 /Columns 300 >>

ID <binary> EI
```

What is refused, and why

tclpdf: "photo" carries 30774 bytes of image data, and ISO 32000-2, 8.9.7 says an inline image should be used only for small images (4096 bytes or less) - tclpdf holds to that line rather than writing a picture the standard advises against; leave -inline off and it goes into the file as an image XObject, which has no such limit and is the cheaper way in for anything this size anyway

tclpdf: "keyed" carries transparency of its own (a transparent colour), so it reaches the file with a Mask or an SMask entry - and an inline image object holds neither, so it would be ignored rather than carried (ISO 32000-2, Table 91 and 8.9.7); leave -inline off, or embed a picture without transparency

The way out is the same in both cases and is named in the message: leave -inline off and the picture goes into the file as an image XObject, which has no limit and no such restrictions. What the package will not do is make that choice silently.