A page to be read back

Ordinary prose, one line per call:

The quick brown fox jumps over the lazy dog.
Every font here is embedded and subsetted, so the codes in the content stream are glyph numbers rather than characters.

A different face, at a different size.

Under a scaled transformation:

Twice the size, drawn at half of it.

Text nobody can read, which is still text:

Two columns, drawn line by line:

A page is a bag of glyphs at positions. Nothing in the file says which of them belong together, or in what order a person would read them.

So an extractor reports what it can prove: one run per show-text operator, with the origin and the total width it advanced by.

Rotated, which keeps its origin and its width:

Thirty degrees off the horizontal.

A second page, laid out by the formatter

XTextFormatter breaks this paragraph into lines and hands each one to DrawString, so the extractor sees one run per line and the lines come back in the order they were drawn. That is reading order here because the layout is a single column, and it is reading order by luck rather than by anything the file records.

Word spacing is the interesting case. The Tw operator adjusts the space between words, and it applies to the single byte 32 and to nothing else - not to a two-byte code whose low byte happens to be 32, which is what every glyph code in a Unicode-encoded font is. That is the trap in the arithmetic, and getting it wrong displaces every run after the first space.

What page one says

ExtractText, printed verbatim. It is a convenience over ExtractRuns and no cleverer than it: runs sharing a baseline are joined - with a space when there is a gap wider than a fifth of the type size, and directly when there is not - and a new baseline starts a new line.

```
A page to be read back
Ordinary prose, one line per call:
The quick brown fox jumps over the lazy dog.
Every font here is embedded and subsetted, so the codes in the
content stream are glyph numbers rather than characters.
A different face, at a different size.
Under a scaled transformation:
Twice the size, drawn at half of it.
Text nobody can read, which is still text:
White on white, and extracted all the same.
Two columns, drawn line by line:
A page is a bag of glyphs at So an extractor reports what
positions. Nothing in the file it can prove: one run per
says which of them belong show-text operator, with the
together, or in what order a origin and the total width it
person would read them. advanced by.
Rotated, which keeps its origin and its width:
Thirty degrees off the horizontal.
```

The two columns came out interleaved

Look for the column lines above: each one is a left-hand line and a right-hand line run together, because they share a baseline and were drawn one after the other. Nothing is wrong. Runs come back in the order they were drawn, which is the order the producer chose and need not be reading order - and grouping runs into columns, paragraphs and a reading sequence is layout analysis, which is a separate piece of work and is deliberately not here.

Invisible is not the same as absent

The white-on-white line from page one is in the text above, because painting a glyph in the colour of the paper does not stop it being a glyph. The one thing the extractor does skip is text render mode 3, which is genuinely invisible and is how the OCR layer under a scan is drawn - a caller asking what the page says usually does not want it twice. There is no such line here to show: XGraphics has no way to draw one, which is why an OCR layer is something this library reads and does not write.

Where page one says it

ExtractRuns gives the origin, the total width and the type size of every run, in user space - the origin at the bottom left of the page, as PDF measures it, rather than at the top left as XGraphics does. The name is the resource name the content stream selected the font by.

x	y	width	size	font	text
50.0	782.0	175.0	16.0	/F0	A page to be read back
50.0	747.0	148.2	9.5	/F0	Ordinary prose, one line per call:
50.0	730.0	180.5	9.0	/F1	The quick brown fox jumps over the lazy dog.
50.0	716.0	254.5	9.0	/F1	Every font here is embedded and subsetted...
50.0	702.0	229.5	9.0	/F1	content stream are glyph numbers rather t...
50.0	674.0	195.7	14.0	/F2	A different face, at a different size.
50.0	637.0	140.3	9.5	/F0	Under a scaled transformation:
50.0	612.0	265.8	18.0	/F1	Twice the size, drawn at half of it.
50.0	567.0	158.9	9.0	/F1	Text nobody can read, which is still text:
50.0	550.0	171.9	9.0	/F1	White on white, and extracted all the same.
50.0	512.0	148.7	9.5	/F0	Two columns, drawn line by line:
50.0	492.0	111.0	9.0	/F1	A page is a bag of glyphs at
300.0	492.0	113.0	9.0	/F1	So an extractor reports what
50.0	478.0	110.0	9.0	/F1	positions. Nothing in the file
300.0	478.0	97.5	9.0	/F1	it can prove: one run per
50.0	464.0	106.0	9.0	/F1	says which of them belong
300.0	464.0	110.5	9.0	/F1	show-text operator, with the
50.0	450.0	108.0	9.0	/F1	together, or in what order a
300.0	450.0	104.5	9.0	/F1	origin and the total width it
50.0	436.0	99.0	9.0	/F1	person would read them.
300.0	436.0	53.5	9.0	/F1	advanced by.
50.0	392.0	203.5	9.5	/F0	Rotated, which keeps its origin and its w...
60.0	282.0	129.0	9.0	/F1	Thirty degrees off the horizontal.

One run per operator, not one box per glyph

A per-glyph box is exact only when every glyph's own advance is known, and reporting an approximate one is worse than reporting none - a caller cannot tell the two apart. A run's origin and total width are exact, and are what most callers actually want.

The scaled line proves the point

The line drawn under a twofold scale reports a size of 18.0 rather than 9, and a width to match. Both are measured through the same matrix, and they have to be: the run is reported in user space, so leaving the current transformation out of one of them would give a width in text space and a size in user space, which disagree with each other.

Page two came back as 7 runs - one per line the formatter laid out.