appligent
DOCUMENT SOLUTIONS

PageWright™

Inline Runs

Styled text inside a paragraph — every inline property, with the JSON that produces it and the line it draws.

Appligent, Inc.
22 East Baltimore Avenue
Lansdowne, PA 19050 U.S.A.

(610) 284-4006
sales@appligent.com
https://www.appligent.com

Version 1.0
October 2026

© 2026 Appligent, Inc. All rights reserved. PageWright is a trademark of Appligent, Inc.

About this manual

This is the **PageWright™ Inline Runs**. PageWright is a PDF page-layout engine: you describe a document as a **JSON spec** - its pages, text, tables, images, charts, forms and structure - and the engine flows it onto pages and writes a tagged, standards-compliant PDF.

Every page of this manual was produced by the engine itself from a JSON spec (`sample/guide-runs.json`), so it doubles as a worked example: the cover, the table of contents, the running heads and folios, the code boxes, the reference tables, and the index and glossary at the back are all ordinary engine features described in these pages.

> Note
>
> Conventions. Monospace text in a red tint marks a JSON key, value, or command - for example `"type": "paragraph"` or `pagewrightapp`. Shaded boxes are complete, valid snippets you can paste into a spec. **Bold** marks the first mention of a defined term; the Glossary collects them.

Contents

About this manual ii
Part I — The model 1
1. What a run is 1
2. The cascade 1
3. Character styles 2
Part II — Setting the type 4
4. Faces, size and color 4
5. Decoration 5
6. Spacing and shape 6
7. Outline and fill 7
8. Raised and lowered 9
Part III — More than text 11
9. Links and cross-references 11
10. Notes 12
11. Language and conditions 13
12. Equations, badges and fields 14
13. Inline graphics 15
Part IV — What a run does not do 19
14. Box properties are accepted and ignored 19
15. Traps 19
A. Every key a run accepts 21
B. What the tags become 21

Part I — The model

1. What a run is

A block of text holds a **`runs`** array, and each run is one stretch of characters plus the inline properties that apply to it. That is how a paragraph carries a bold word, a link, a footnote marker, a superscript, a language span or an equation without being broken into separate blocks.

`runs` is read by `paragraph`, `heading`, `callout`, a list item, a table cell, a caption written in run form, and a running header/footer slot. One decoder reads them all, so a run behaves identically wherever it appears. A run is either a bare string or an object.

A first run

```
{
  "type": "paragraph",
  "runs": [
    {"text": "Plain text, then "},
    {"text": "a bold word", "style": {"weight": "bold"}},
    {"text": ", then a "},
    {
      "text": "link",
      "link": "https://appligent.com",
      "style": {"color": "#1a5fb4", "underline": true}
    },
    {"text": "."}
  ]
}
```

Renders as Plain text, then **a bold word**, then a link.

> Tip
>
> `runs` also accepts a single run object instead of an array, which is convenient when a block has exactly one.

2. The cascade

A run's style resolves over its block's, which resolves over the document's `defaultStyle`. **Text properties inherit; box properties belong to the block that sets them.** A run that names nothing keeps everything it was given.

Inheriting, then overriding

```
{
  "type": "paragraph",
  "runs": [
    {"text": "This sentence is set in the block's style, and "},
    {
      "text": "this clause overrides two properties of it",
      "style": {"italic": true, "color": "#a4133c"}
    },
    {"text": ", then the sentence continues."}
  ]
}
```

Renders as This sentence is set in the block's style, and *this clause overrides two properties of it*, then the sentence continues.

3. Character styles

A run's `style` may be an **inline object** or the **name of an entry in the document's `styles` table** - which is what a character format is here. `basedOn` derives one entry from another, and an object may carry `basedOn` plus its own overrides, so a house style lives in one place and a document's runs refer to it.

A named character style

The `styles` table is a document-level key; the runs below name its entries.

```
{
  "styles": {
    "Emphasis": {"italic": true, "color": "#a4133c"},
    "Strong": {
      "basedOn": "Emphasis",
      "italic": false,
      "weight": "bold",
      "color": "#1a4f7a"
    }
  },
  "blocks": [
    {
      "type": "paragraph",
      "runs": [
        {"text": "ordinary text, "},
        {"text": "Emphasis", "style": "Emphasis"},
        {"text": ", "},
        {"text": "Strong", "style": "Strong"},
        {"text": ", and "},
        {
          "text": "Emphasis at 14pt",
          "style": {"basedOn": "Emphasis", "fontSize": 14}
        },
        {"text": "."}
      ]
    }
  ]
}
```

Renders as ordinary text, *Emphasis*, **Strong**, and *Emphasis at 14pt*.

Part II — Setting the type

4. Faces, size and color

A different face mid-line

Every face a run names is embedded and subset into the finished PDF.

```
{
  "type": "paragraph",
  "runs": [
    {"text": "the body serif, then "},
    {"text": "the display sans", "style": {"fontFamily": "Myriad Pro"}},
    {"text": ", then "},
    {"text": "the code face", "style": {"fontFamily": "Courier Std"}},
    {"text": "."}
  ]
}
```

Renders as the body serif, then the display sans, then `the code face`.

Size

```
{
  "type": "paragraph",
  "runs": [
    {"text": "body size, "},
    {"text": "larger", "style": {"fontSize": 14}},
    {"text": ", "},
    {"text": "smaller", "style": {"fontSize": 7}},
    {"text": "."}
  ]
}
```

Renders as body size, larger, smaller.

Weight and italic

`"bold": true` is shorthand for `"weight": "bold"`.

```
{
  "type": "paragraph",
  "runs": [
    {"text": "plain, "},
    {"text": "bold", "style": {"weight": "bold"}},
    {"text": ", "},
    {"text": "italic", "style": {"italic": true}},
    {"text": ", "},
    {"text": "both", "style": {"weight": "bold", "italic": true}},
    {"text": "."}
  ]
}
```

Renders as plain, **bold**, *italic*, ***both***.

Color

Any of the engine's four color spaces may be used here - RGB, gray, CMYK or a spot ink. `"none"` fills nothing, leaving letters that show only their stroke.

```
{
  "type": "paragraph",
  "runs": [
    {"text": "inherited, "},
    {"text": "crimson", "style": {"color": "#a4133c"}},
    {"text": ", "},
    {"text": "navy", "style": {"color": "#1a4f7a"}},
    {"text": "."}
  ]
}
```

Renders as inherited, crimson, navy.

5. Decoration

Underlines, strike and overline

`underline` takes `true` or one of `"none"`, `"single"`, `"double"`, `"numeric"` - FrameMaker's set. `"numeric"` is carried distinctly and draws as single today.

```
{
  "type": "paragraph",
  "runs": [
    {"text": "single", "style": {"underline": true}},
    {"text": ", "},
    {"text": "double", "style": {"underline": "double"}},
    {"text": ", "},
    {"text": "strike", "style": {"strike": true}},
    {"text": ", "},
    {"text": "overline", "style": {"overline": true}},
    {"text": "."}
  ]
}
```

Renders as single, double, ~~strike~~, overline.

Small capitals

Synthesized from the regular face, and the real words are preserved in `/ActualText`, so search and a screen reader still read what you wrote. A face with true small capitals can be asked for instead with `"features": ["smcp"]`.

```
{
  "type": "paragraph",
  "runs": [
    {"text": "Small Capitals", "style": {"smallCaps": true}},
    {"text": " beside ordinary text."}
  ]
}
```

Renders as SMALL CAPITALS beside ordinary text.

6. Spacing and shape

Letter spacing

`tracking` is an alias. Word spacing is a BLOCK property - see *What a run does not do*.

```
{
  "type": "paragraph",
  "runs": [
    {"text": "tracked out", "style": {"letterSpacing": 1.4}},
    {"text": " and back to normal."}
  ]
}
```

Renders as t r a c k e d o u t and back to normal.

Condensing and extending

`stretch` is a percentage of normal width - FrameMaker's FP_Stretch, the PDF `Tz` operator. Letterforms and advances scale; the size and the baseline do not.

```
{
  "type": "paragraph",
  "runs": [
    {"text": "condensed to 70", "style": {"stretch": 70}},
    {"text": " - "},
    {"text": "extended to 130", "style": {"stretch": 130}},
    {"text": "."}
  ]
}
```

Renders as condensed to 70 - extended to 130.

OpenType features

`liga`, `dlig`, `onum`, `lnum`, `tnum`, `pnum` and `smcp`, on an embedded face that defines them. A face without the feature is unchanged rather than faked - the two lines above are identical unless this book's face carries both figure sets.

```
{
  "type": "paragraph",
  "runs": [
    {"text": "oldstyle 0123456789", "style": {"features": ["onum"]}},
    {"text": " against "},
    {"text": "lining 0123456789", "style": {"features": ["lnum"]}},
    {"text": "."}
  ]
}
```

Renders as oldstyle 0123456789 against lining 0123456789.

7. Outline and fill

A stroked outline

A PDF stroke straddles the outline, so it always thickens the type a little; a width tuned for display sizes is far too heavy in body text.

```
{
  "type": "paragraph",
  "runs": [
    {
      "text": "outlined letters",
      "style": {"strokeWidth": 0.4, "strokeColor": "#a4133c", "weight": "bold"}
    },
    {"text": " beside plain ones."}
  ]
}
```

Renders as **outlined letters** beside plain ones.

Paint through the glyphs

`textFill` paints a gradient or an image through the letterforms. The glyphs become a clipping path; the run stays real text in the structure tree - selectable, searchable and tagged.

```
{
  "type": "paragraph",
  "runs": [
    {
      "text": "a gradient painted straight through the letterforms",
      "style": {
        "weight": "bold",
        "fontSize": 17,
        "textFill": {
          "type": "gradient",
          "gradient": {
            "type": "linear",
            "angle": 0,
            "stops": [
              {"offset": 0, "color": "#1b3a5c"},
              {"offset": 1, "color": "#a4133c"}
            ]
          },
          "extent": "line"
        }
      }
    }
  ]
}
```

Renders as **a gradient painted straight through the letterforms**

Caution

The ramp maps across the COLUMN, not across your words. `extent` chooses which box - `line` restarts the ramp on every line, `block` runs one ramp down a wrapped heading - and either way it is the measured box, never the inked width of the run.

So a SHORT run shows only the beginning of the gradient. The specimen above runs most of the way across the column and reaches the crimson; at a third of that width it reads as flat navy, with no hint that anything was wrong. Size the run to the box you want the ramp to cross.

Note

The six `textEffect` values - `shadow`, `lift`, `echo`, `glow`, `neon` and `background` - are run properties too, and they are drawn in the `text-effects` sample rather than here. PDF has no glyph-blur operator, so an effect is the run STAMPED AGAIN up to twenty-five times: printing one on this page would add two thousand overprinting glyph pairs to a book whose gate stages exist to find exactly that.

`background` does appear later, where it earns its place by contrast.

8. Raised and lowered

Superscript and subscript

Both flags MOVE the run and neither RESIZES it, which is why each example above also sets `fontSize`. Set on the same run, `subscript` is ignored.

```
{
  "type": "paragraph",
  "runs": [
    {"text": "x"},
    {"text": "2", "superscript": true, "style": {"fontSize": 7}},
    {"text": " and H"},
    {"text": "2", "subscript": true, "style": {"fontSize": 7}},
    {"text": "O."}
  ]
}
```

Renders as x^2 and H_2O.

Moving a run off its baseline

`baselineShift` moves a run by that many points, positive up and negative down, on top of where `superscript` or `subscript` puts it, or on its own. The first mark is lifted 2 pt above the second, and the 2 set 1 pt lower. Only the ink moves: the line keeps its height.

```
{
  "type": "paragraph",
  "runs": [
    {"text": "PageWright"},
    {
      "text": "™",
      "superscript": true,
      "baselineShift": 2,
      "style": {"fontSize": 6}
    },
    {"text": " beside PageWright"},
    {"text": "™", "superscript": true, "style": {"fontSize": 6}},
    {"text": " and H"},
    {
      "text": "2",
      "subscript": true,
      "baselineShift": -1,
      "style": {"fontSize": 7}
    },
    {"text": "O."}
  ]
}
```

Renders as PageWright™ beside PageWright™ and H_2O.

Part III — More than text

9. Links and cross-references

An external and an internal link

A `link` value beginning `#` is an internal jump to a block's `id`; anything else is a URL. **A PDF link annotation draws nothing of its own**, so a link you do not style is invisible - the two above carry `color` and `underline` to prove the point. Setting `linkStyle` at document level does the same thing once, for every link in the book.

```
{
  "type": "paragraph",
  "runs": [
    {
      "text": "an external link",
      "link": "https://appligent.com",
      "style": {"color": "#1a5fb4", "underline": true}
    },
    {"text": " and an "},
    {
      "text": "internal one",
      "link": "#links",
      "style": {"color": "#1a5fb4", "underline": true}
    },
    {"text": " to this chapter."}
  ]
}
```

Renders as an external link and an internal one to this chapter.

A live cross-reference

`ref` names a block's `id` and `as` chooses what to render: `page`, `number`, `label`, `title` or `full`. The run's own text is ignored - the engine composes it, and links the reference to its target. `number` and `label` need a numbering scheme and degrade to the page without one. `title` and `full` never do: `full` names the target by its label where there is a number and by its title where there is not, so it reads as a sentence in any document.

```
{
  "type": "paragraph",
  "runs": [
    {"text": "This chapter is called "},
    {"ref": "links", "as": "title"},
    {"text": ", on page "},
    {"ref": "links", "as": "page"},
    {"text": "."}
  ]
}
```

Renders as This chapter is called 9. Links and cross-references, on page 11.

Custom reference wording

`refTemplate` composes the text yourself from `{number}`, `{word}`, `{label}`, `{title}` and `{page}`.

```
{
  "type": "paragraph",
  "runs": [
    {"text": "see "},
    {"ref": "cascade", "refTemplate": "{title} on p.{page}"},
    {"text": "."}
  ]
}
```

Renders as see 2. The cascade on p.1.

10. Notes

A footnote

The marker is drawn where the run sits and the note is gathered to the page foot. `endnote` does the same into the next `endnotes` block, and `sidenote` into the reserved side band (`pageSetup.sideHeadWidth`), with no marker at all.

```
{
  "type": "paragraph",
  "runs": [
    {"text": "A sentence with a note"},
    {
      "text": "",
      "footnote": "The marker is raised here; the note itself is set at the foot of
        this page, numbered and linked."
    },
    {"text": " that carries on afterwards."}
  ]
}
```

Renders as A sentence with a note[1] that carries on afterwards.

A citation

`cite` names one or more keys from the document's `bibliography` block; the marker's form comes from that block's style, numbered by order of first use, and links forward to the entry and back again. The run's own text is ignored. The `bibliography` sample shows the whole apparatus.

```
{
  "type": "paragraph",
  "runs": [
    {"text": "as argued elsewhere "},
    {"cite": "knuth1984", "citePages": "pp. 12-14"},
    {"text": "."}
  ]
}
```

Renders as as argued elsewhere .

11. Language and conditions

A language span

`lang` produces a `<Span>` carrying `/Lang`, which is what a screen reader switches voices on, and it also selects the hyphenation rules for that run.

1. The marker is raised here; the note itself is set at the foot of this page, numbered and linked.

```
{
  "type": "paragraph",
  "runs": [
    {"text": "English, then "},
    {"text": "Bonjour le monde", "lang": "fr"},
    {"text": ", marked fr."}
  ]
}
```

Renders as English, then Bonjour le monde, marked fr.

Conditional text

A run carrying a `condition` is dropped unless the document shows that tag. Turn `conditionIndicators` on for an author view, where hidden runs are struck through and shown in their tag's color instead of being removed.

```
{
  "conditions": {"draft": true, "internal": false},
  "blocks": [
    {
      "type": "paragraph",
      "runs": [
        {"text": "shipped in every build; "},
        {"text": "[draft only]", "condition": "draft"},
        {"text": " and the sentence closes."}
      ]
    }
  ]
}
```

Renders as shipped in every build; [draft only] and the sentence closes.

12. Equations, badges and fields

An inline badge

`badge` carries the label text - the run's `text` is not used - and `badgeStyle` carries the box. A radius of 999 or more renders a full pill. The fill is an artifact, so a screen reader reads only the label.

```
{
  "type": "paragraph",
  "runs": [
    {"text": "a release marked "},
    {
      "badge": "NEW",
      "style": {"color": "#ffffff", "fontSize": 7.5},
      "badgeStyle": {"background": "#1a4f7a", "radius": 999, "padX": 4, "padY": 1.5}
    },
    {"text": " in running text."}
  ]
}
```

Renders as a release marked NEW in running text.

An **inline equation** is a run carrying `mathml`, `mathSvg` and `mathAlt`: the MathML is attached to the PDF as an associated file, the SVG is drawn on the baseline, and the alt text is what is read aloud. An **inline form field** is a run carrying `field` plus its `name`, `width` and `height`. Both are shown live in the corpus - `math-equations` and `forms-inline` - because each needs more setup than a single line can carry.

```
{
  "type": "paragraph",
  "runs": [
    {
      "text": "the identity "
    },
    {
      "mathml":
        "<math><mi>E</mi><mo>=</mo><mi>m</mi><msup><mi>c</mi><mn>2</mn></msup></math>",
      "mathSvg": "<svg ...>",
      "mathAlt": "E equals m c squared"
    },
    {
      "text": " set in the line"
    }
  ]
}
```

13. Inline graphics

A run can carry a **graphic** instead of characters: `image` names a file, `svg` carries the markup directly, and `alt` says what it is. It draws on the baseline like a letter, which is what makes it an icon, a logo, a signature or a symbol no font has - the inline counterpart of the `image` block, and the run form of an anchored graphic.

An icon on the line

The run draws the graphic and nothing else - any `text` it carries is used as the description when `alt` is absent.

```
{
  "type": "paragraph",
  "runs": [
    {"text": "Status "},
    {
      "svg": "<svg xmlns=\"http://www.w3.org/2000/svg\" viewBox=\"0 0 12 12\"><path
        d=\"M2 6.5 L5 9.5 L10 3\" stroke=\"#1a7f37\" stroke-width=\"2\"
        fill=\"none\"/></svg>",
      "alt": "a check mark"
    },
    {"text": " approved, "},
    {
      "svg": "<svg xmlns=\"http://www.w3.org/2000/svg\" viewBox=\"0 0 12 12\"><path
        d=\"M3 3 L9 9 M9 3 L3 9\" stroke=\"#b42318\" stroke-width=\"2\"
        fill=\"none\"/></svg>",
      "alt": "a cross"
    },
    {"text": " rejected."}
  ]
}
```

Renders as Status ✔ approved, ✖ rejected.

Size comes from the text, not from the graphic. Left alone, a graphic is one em tall and takes its width from the source's own aspect, so a 512-pixel logo does not open a 512-point line, and an icon follows the type when the style around it changes. `height` makes it bigger and the line grows to hold it; giving **both** `width` and `height` sets a box of another shape on purpose. An `image` is stretched into it; an `svg` keeps its own shape, centred in it, unless its root `<svg>` says `preserveAspectRatio="none"`.

Three sizes of the same mark

The first takes the run's font size. The second sets a height and keeps the aspect. The third sets both, and its svg says preserveAspectRatio="none", so the mark is drawn to the box it was given.

```
{
  "type": "paragraph",
  "runs": [
    {"text": "inherited "},
    {
      "svg": "<svg xmlns=\"http://www.w3.org/2000/svg\" viewBox=\"0 0 12 12\"><path
        d=\"M2 6.5 L5 9.5 L10 3\" stroke=\"#1a7f37\" stroke-width=\"2\"
        fill=\"none\"/></svg>",
      "alt": "a check mark at the text size"
    },
    {"text": "  ·  at 18pt "},
    {
      "svg": "<svg xmlns=\"http://www.w3.org/2000/svg\" viewBox=\"0 0 12 12\"><path
        d=\"M2 6.5 L5 9.5 L10 3\" stroke=\"#1a7f37\" stroke-width=\"2\"
        fill=\"none\"/></svg>",
      "height": 18,
      "alt": "a larger check mark"
    },
    {"text": "  ·  stretched "},
    {
      "svg": "<svg xmlns=\"http://www.w3.org/2000/svg\" viewBox=\"0 0 12 12\"
        preserveAspectRatio=\"none\"><path d=\"M2 6.5 L5 9.5 L10 3\" stroke=\"#1a7f37\"
        stroke-width=\"2\" fill=\"none\"/></svg>",
      "width": 40,
      "height": 10,
      "alt": "a check mark stretched wide"
    }
  ]
}
```

Renders as inherited ✓ · at 18pt ✓ · stretched ✓

```
{
  "type": "paragraph",
  "runs": [
    {
      "text": "signed "
    },
    {
      "image": "signature.png",
      "height": 24,
      "alt": "the signature of A. Reviewer"
    },
    {
      "text": " on this date"
    }
  ]
}
```

Caution

A graphic needs `alt`. It draws no glyphs, so `/Alt` is the only thing a screen reader can be given for it - and unlike an equation there is no MathML to fall back on. Without one the engine tags the placeholder `"graphic"`, says so on stderr, and `-wcag` reports it under 1.1.1.

Tip

A graphic is tagged `<Figure>`; an equation, which is a run carrying `mathml` or `mathSvg`, is tagged `<Formula>`. That is decided by which keys the run carries, so `svg` beside `mathml` is the equation's own SVG rather than a second picture.

Part IV — What a run does not do

14. Box properties are accepted and ignored

A run takes the same `style` object a block takes, so a **box** property can be written on a run - and it does nothing at all. A run is not a box: it has no padding, no border and no background of its own.

The one that catches everybody

To tint words behind the type, reach for `"textEffect": "background"`.

```
{
  "type": "paragraph",
  "runs": [
    {"text": "background on a run", "style": {"background": "#f7e8d4"}},
    {"text": " draws no tint; "},
    {
      "text": "textEffect background",
      "style": {"textEffect": "background"}
    },
    {"text": " is the one that does."}
  ]
}
```

Renders as background on a run draws no tint; textEffect background is the one that does.

written on a run	what happens
`background`, `padding`, `border`, `radius`, `stripe`, `accentBar`, `shadow`, `minHeight`	accepted, ignored - box properties
`align`, `lineHeight`, `firstLineIndent`, `tabs`, `spaceBefore`, `spaceAfter`, `fitWidth`	accepted, ignored - they describe a whole block
`wordSpacing`, `direction`	text properties, but read from the BLOCK; set them there

15. Traps

> Caution
>
> **A block written with `text:` instead of `runs:` carries no run keys at all.** `link`, `footnote`, `endnote`, `sidenote`, `cite` and `citePages` belong on a run; written on the block itself - in either spelling - they are ignored, and the engine says so on stderr: `"footnote" belongs on a run, not on the block`.
>
> To mark a whole paragraph, give it one run: `"runs": [ { "text": "...", "footnote": "..." } ]`.

> **Caution**
>
> **One run is one atom.** `svg` and `content` beside `mathml` or `mathSvg` are the EQUATION's own markup, not a second picture - the math keys claim the run, and the same spellings mean a graphic only when no math key is present.

> **Caution**
>
> **An inline graphic is sized from the text.** Unsized it is one em tall with its width from the source's aspect. Reach for `height` to make it bigger; give both `width` and `height` only when overriding the aspect is what you mean.

> **Caution**
>
> **An unknown key costs the words.** `runs` is validated with `unevaluatedProperties: false`, so a misspelled key makes the whole run invalid rather than merely unstyled - validate against `aplayout.schema.json` before a render.

A. Every key a run accepts

key	what it does
`text`	the characters this run contributes
`style`	an inline style object, or a name from the document `styles` table
`superscript`, `subscript`	raise or drop the run - neither resizes it
`link`	a URL, or `#id` for an internal jump
`ref`, `as`, `refTemplate`	a live cross-reference to a block's `id`
`footnote`, `endnote`, `sidenote`	a note at the page foot, the endnotes block, or the side band
`cite`, `citePages`	a citation into the document's `bibliography`
`lang`	a language span - `<Span>` with `/Lang`, and the hyphenation rules
`condition`	conditional text at run level
`mathml`, `mathSvg`, `mathAlt`	an inline equation and its spoken reading
`image` or `svg`, + `alt`, `width`, `height`	an inline graphic on the baseline: an icon, a logo, a signature
`badge`, `badgeStyle`	an inline pill: the label, and the box behind it
`field`, `name`, `value`, `options`, `tooltip`, `width`, `height`	a live AcroForm widget in running text

B. What the tags become

inline feature	structure it produces
`lang`	`<Span>` carrying `/Lang`
`link` (URL)	`<Link>` with a `/URI` annotation and `/OBJR`
`link` (`#id`), `ref`	`<Link>` with a GoTo destination
`footnote`, `sidenote`, `endnote` body	`<Note>` - the footnote's `/ID` satisfies PDF/UA 7.9
`endnote` and `cite` markers	`<Link>` to the note or entry, and back
`mathml` / `mathSvg`	`<Formula>` with `/Alt` and the MathML as an associated file
`image` / `svg`	`<Figure>` with `/Alt` (the placeholder `graphic`, and a warning, when absent)
`field`	`<Form>` and the widget annotation
`smallCaps`	`<Span>` with `/ActualText` restoring the real words
`badge`	the pill's fill is an artifact; the label is ordinary text

Two deliberate absences: the **footnote marker** in body text carries no tag and no link (the note body does), and cross-references and citations get no `<Reference>` wrapper, even under PDF/UA-2.