The Triplicate: A Data-Driven Large-Format Newspaper Layout Engine

Designed multi-column page layout as a reproducible-research exemplar

Daniel Ari Friedman
Active Inference Institute
daniel@activeinference.institute
ORCID: 0000-0001-6232-9096

June 1, 2026

Contents

1 Abstract

We present `template_newspaper`, a pure-Python engine that renders a complete twelve-page, large-format newspaper to a print-ready PDF from structured YAML content. The exemplar edition is *The Triplicate*, a homage to the historic newspaper of Crescent City, California (founded 1879). The engine demonstrates designed multi-column page layout — nameplate and ears, spanning headlines, flowing column frames with an optional rail, drop caps, pull quotes, ruled modular boxes, data tables, halftone "engraving" figures and running folios — while keeping a strict separation between *content* (data) and *engine* (code): a new title is a data edit, never a code change. The layout strategy is a hybrid in which fixed *furniture* is drawn directly on the canvas to establish where the column grid begins, after which body copy flows through ReportLab frames that split paragraphs across columns automatically. The project obeys the research-template monorepo contract and is discovered and executed by the same orchestration pipeline as its code- and prose-focused siblings.

2 Introduction

Most reproducible-research tooling treats a document as a linear stream: a manuscript is sections of prose, rendered top to bottom. A newspaper is the opposite — a deliberately *non-linear*, spatial artifact, in which a reader's eye is guided by a hierarchy of headline sizes, by rules and gutters, by the placement of a photograph and the interruption of a pull quote. Building one programmatically is therefore an unusually demanding test of a layout system, and an unusually legible demonstration when it succeeds.

`template_newspaper` exists to be that demonstration within the research template. It is the third canonical exemplar, joining `template_code_project` (a computational pipeline) and `template_prose_project` (a manuscript-quality pipeline). All three share one directory contract and one orchestration pipeline; they differ only in what they render.

The design follows a single principle: **content is data, the engine is the press.** An edition is a small set of YAML files under `content/` — a masthead manifest and one file per page, each declaring its stories, boxes and figures. The engine in `src/newspaper/` reads that data and renders it; it contains no copy and, outside a single typography module, names no fonts. Producing a different paper — a school gazette, a conference daily, a neighborhood broadsheet — is an edit to the data, not the code.

The remainder of this manuscript describes the engine's architecture (Section 2), its typography and figure generation (Section 3), and how the artifact is produced and verified reproducibly (Section 4).

3 Engine Architecture

The engine is eight small modules, each with one responsibility.

`geometry` is pure arithmetic over points: the page trim, its margins, and a `ColumnGrid` that partitions a region into equal columns with gutters and reports the centre of each gutter (where a hairline rule is drawn). It imports no rendering library, which makes the column mathematics trivially testable.

`content` defines the typed object graph — `Edition`, `Page`, `Story`, `Box`, `Figure`, `Block` — and the strict YAML loaders that build it. The loaders are forgiving about *shape* (a body element may be a bare string or a tagged mapping) but strict about *structure*: an unknown page template or box kind raises an error naming the offending value and the allowed set.

`config` is the strict render configuration (trim size, default column count, rail width, gutter), validated the same way.

`components` builds the flowables that live *inside* a column: a story's headline, byline and body; a raised drop cap; a pull quote; a ruled box; a data table; a figure with its cutline. `furniture` draws the elements that are *placed* rather than flowed: the nameplate and its ears, the section banner, the folio, the hairline column rules, and the spanning lead headline.

`layout` is the bridge. For each page it draws the furniture (which returns the y-coordinate where the column grid begins), constructs the column frames — including an optional narrow rail and the room reserved for a spanning headline — and pours the flowables into the frames with ReportLab's `Frame.addFromList`, which splits paragraphs across columns automatically. Content that does not fit is returned and reported as an *over-set* page rather than silently dropped.

`engine` ties it together: register fonts, build the stylesheet, open a canvas at the configured trim, render each page, and write the PDF plus a machine-readable render report.

This **drawn-furniture / flowed-body** split is the key architectural choice. A pure document-template approach cannot place a headline that spans several columns above body text that flows beneath it; drawing the furniture first, and beginning the column frames below it, makes spanning headlines, standing boxes and automatic text flow coexist on the same page.

4 Typography and Figures

A newspaper's voice is its type. The engine registers, when present, the elegant text faces a real paper would license — Didot for the nameplate and display headlines, Georgia for body text, and a grotesque sans for kickers and labels — and falls back to ReportLab's always-available base-14 fonts when a face is missing. Registration is best-effort and never fatal: a checkout on a machine without the macOS supplemental fonts still renders a structurally identical paper in Times and Helvetica. Only the typography module names a font; every other module refers to four logical roles (display, body, sans, and their bold and italic variants), so a change of typeface touches one file.

A subtle lesson is recorded in the code: the bold weight inside a `.ttc` font collection is at a face-specific subfont index. Didot's collection orders its faces Regular, Italic, Bold; an early render produced *italic* headlines because the bold index was assumed to be one rather than two. The fix — and the comment that now guards it — came from inspecting the rendered raster, not the source.

Figures are generated in the monochrome idiom of newsprint by two means. Photo- graphic elements — the harbor, the lighthouse, the redwoods — are procedural grayscale scenes drawn with Pillow and passed through a forty-five-degree halftone screen, so they read as classic dot-screened newspaper photographs. Data graphics — a seven-day forecast, a tide curve, a Dungeness-crab landings bar chart — are rendered with Matplotlib in black and gray with serif labels, so they sit naturally beside the type rather than fighting it. A missing image file degrades to a clearly labelled placeholder, never a crash.

5 Reproducibility

The artifact is produced by a deterministic, three-step chain that the repository pipeline runs as Stage-02 analysis scripts, in lexicographic order: `00_preflight.py` confirms the dependencies import and the edition loads and validates; `10_generate_figures.py` writes the halftone scenes and charts to `../figures/`; and `20_render_newspaper.py` renders the twelve pages to `output/pdf/the-triplicate.pdf` and emits a machine-readable render report (`output/data/render_report.json`) plus a human summary. The render is deterministic — the canvas is opened in ReportLab's `invariant` mode and the figures are drawn from fixed procedural recipes — so the same content yields the same bytes.

Correctness is established two ways. A `pytest` suite exercises the pure logic (geometry, configuration, content loaders, typography, figure generation) and renders the real edition end-to-end, asserting that the output is a valid PDF of exactly twelve pages at the correct trim with no over-set page. Because the remaining correctness of a *layout* is visual, the development loop rasterizes each page and inspects it; the project's agent guidance makes that visual check mandatory after any layout change. The engine is type-checked with mypy and linted with ruff, and the test suite reports roughly ninety-five percent line coverage.

Every quantitative claim in this project — the page count, the trim dimensions, the figure count — is recorded in `data/claim_ledger.yaml` against the artifact that substantiates it, so a reviewer can trace each number to its source.

6 References

The bibliography is maintained in `references.bib`. Key sources:

- ReportLab Ltd. *ReportLab PDF Library — User Guide.* The platypus flowable and frame model underpins this engine's column flow [ReportLab Ltd., 2024].
- Bringhurst, R. *The Elements of Typographic Style.* The measure, scale and rule conventions that inform the stylesheet [Bringhurst, 2013].
- Wong, B. "Points of view: Color blindness." *Nature Methods* (2011). The basis for accessible, here monochrome, figure palettes [Wong, 2011].
- García, M. *Contemporary Newspaper Design.* Grid, modular layout and the rail/well distinction [García, 1987].

References

Robert Bringhurst. *The Elements of Typographic Style*. Hartley & Marks, 4 edition, 2013.

Mario R. García. *Contemporary Newspaper Design*. Prentice Hall, 1987.

ReportLab Ltd. *ReportLab PDF Library — User Guide*, 2024. URL https://www.reportlab.com/docs/reportlab-userguide.pdf.

Bang Wong. Points of view: Color blindness. *Nature Methods*, 8(6):441, 2011. doi: 10.1038/nmeth.1618.