Zen Working Group
Request for Discussion and Implementation Record

ZDS 0015
Implementation Specification

Browser WebAssembly and the zenfmt Project Site

STATE
committed

CREATED
2026-08-08

LAST UPDATED
2026-08-09

INTENDED STATUS
Committed

AUTHORS
Vikrant Rathore
Ronak Rathore (assistance)

DISCUSSION

The implementation blueprint for the 0.2.0 browser WASM release, conversion playground, project website, HTML book and ZDS, and public benchmark dashboard

LABELS
wasm, browser, website, documentation, benchmark, release

Status of This Memo

This document is an internal Zen discussion record authored in Typst and tracked in git. It intentionally follows RFC-style structure so the design scope, rationale, trade-offs, and operational constraints remain explicit. Documents that still use the placeholder number XXXXX are provisional drafts. Numbered ZDS documents are part of the project record.

Contents

142. Abstract

zenfmt 0.1.0 delivered the native command-line application and the Python library specified by ZDS 0014. The next public surface is the browser. Release 0.2.0 will compile the same Zig conversion engine directly to WebAssembly, publish a small versioned browser adapter, and load that module on the zenfmt GitHub Pages site. A visitor drops a supported document, the bytes move to a dedicated Web Worker, zenfmt converts them locally, and an adjacent read-only code window displays the Markdown. The document is never uploaded and the Markdown is never interpreted as HTML.

The website is not merely a demo. It becomes the project front door at `https://insanai.github.io/zenfmt/`. Pandoc showed how useful a universal document converter can be; zenfmt is a small attempt to explore that idea with an engine written in Zig. The site borrows useful interaction ideas from AnyDoc and a restrained editorial layout from Notion, without borrowing their voice or identity. It remains keyboard accessible, privacy preserving, and free of a server runtime. It presents the converter first, an auditable benchmark dashboard second, and clear routes into the zenfmt book, Zen Discussion records, downloads, and GitHub. The default theme follows the operating system. The visitor can choose light, dark, or system, and the preference stays on that device.

The book and every ZDS remain authored in Typst. Their shared sources produce archival PDFs and a new multi-page HTML documentation site with the navigation quality associated with Sphinx: book tree, local table of contents, search, permalinks, previous/next navigation, and stable URLs. The book and ZDS are also contextual help. A visitor can move from a failed conversion to the relevant quick explanation, then to the book chapter, then to the governing design record without guessing where documentation lives.

This record is the normative implementation blueprint for release 0.2.0. It defines the WASM target and ABI, browser API, worker and memory model, site information architecture, interaction and visual systems, HTML/PDF publishing, benchmark methodology, accessibility, security, build graph, CI, release artifacts, rollout, and acceptance gates. No part of that implementation is created by this record; moving the record to `committed` requires all of it.

143. Introduction

A document converter is unusually well suited to an in-browser experience. The input already exists on the visitor's device, the useful output is usually text, and zenfmt's engine is native Zig without a service dependency. Running the engine as WebAssembly removes installation from the first experience and turns "does zenfmt handle my file?" into a question answered by the file itself. It also creates a strong privacy boundary: GitHub Pages serves static assets, while conversion occurs inside the visitor's browser.

The design must not confuse immediate usefulness with shallowness. New users need one obvious action and one obvious result. Evaluators need format coverage, limitations, benchmarks, raw data, and security facts. Contributors need the book, ZDS records, APIs, build instructions, and source. Those are different cognitive modes and should not compete in one undifferentiated page.

This record uses **System 1** and **System 2** as interface-design shorthand, not as a claim that a website can classify human cognition. The System 1 path is fast recognition: a visible drop target, strong hierarchy, plain status text, safe defaults, and copy/download actions. The System 2 path supports deliberate inspection: advanced options, structured diagnostics, benchmark methodology, the book, ZDS records, and implementation references. Progressive disclosure connects the paths. Nothing safety-critical is hidden merely because it is detailed.

Steve Krug's "don't make me think" principle supplies the practical test. A first-time visitor should not have to infer whether the site uploads files, which pane is input, whether output is editable, where help lives, or whether a speed claim is comparing like with like. Labels and layout answer those questions before prose does.

143.1. Relationship to existing records

- ZDS 0002 remains authoritative for engine composition, format detection, limits, manifests, reports, and the CLI.
- ZDS 0013 remains authoritative for the layered document IR and writer lowering.
- ZDS 0014 remains authoritative for the Python API and unified release version. Its artifact-parity and Elm-style diagnostic rules also apply to the browser boundary.
- ZDS 0001 remains authoritative for ZDS lifecycle and numbering. This record extends the publishing implementation but does not change the lifecycle.

When this record conflicts with conversion semantics in ZDS 0002 or ZDS 0013, those engine records win. This record decides how the browser invokes and presents those semantics.

143.2. Normative language and completion

Must and **required** identify binding release requirements. **Should** identifies a default that can change only with recorded evidence. **May** identifies an allowed choice. References to future work are explicitly outside 0.2.0.

The ZDS is a record of intended implementation. "No implementation in this document" means the change is not being implemented in this authoring task; it does not make implementation a non-goal. The record can move to `committed` only after every deliverable and acceptance gate below exists in the repository and in the published release.

144. Terminology and Scope

- **WASM module**: the `wasm32-freestanding` executable produced directly by Zig from the default zenfmt bundle;
- **browser adapter**: the small standards-based ES module that loads WASM, validates versions, marshals bytes, and exposes the public browser API;
- **worker adapter**: the ES module running conversion inside a dedicated Web Worker so parsing never blocks the page's interaction thread;
- **playground**: the converter workspace on the project homepage;
- **artifact ensemble**: Markdown bytes, embedded resources, canonical manifest, and ordered reports from one memory conversion;
- **site shell**: shared header, navigation, search, theme, footer, metadata, and accessibility structure across the homepage, book, and ZDS pages;
- **book site**: chapter-oriented HTML generated from `docs/book.typ` and `docs/book/` while preserving the PDF edition;
- **ZDS site**: registry-driven HTML and PDF output generated from `docs/zds/records/`;
- **reference benchmark**: a checked-in, reproducible run on a named machine or pinned CI environment, never measurements silently collected from visitors;
- **native lens**: process and installed-library comparisons on a native host;
- **browser lens**: WASM startup and warm conversion comparisons in a pinned browser;
- **System 1 path**: the immediate convert, understand, recover, copy, or download path;
- **System 2 path**: the deliberate inspect, compare, learn, verify, and contribute path.

In scope:

- a Zig-native browser WebAssembly build of the complete default reader and Markdown-writer bundle;
- a versioned low-level WASM ABI and ergonomic browser JavaScript API;
- worker-based local document conversion on the GitHub Pages homepage;
- the zenfmt site design, responsive layouts, and theme selector;

- book and ZDS HTML with one shared documentation shell and stable routes;
- linked, downloadable book and ZDS PDFs;
- contextual help that connects the converter, book, and ZDS;
- a transparent native and browser benchmark dashboard comparing zenfmt, AnyDoc, and Pandoc where a fair comparison is available;
- root Zig build integration and uv-managed Python site tooling;
- browser, accessibility, security, performance, documentation, and release tests;
- GitHub Pages build and deployment through GitHub Actions;
- a checksummed WASM browser bundle in the unified 0.2.0 GitHub release;
- regenerated 0.2.0 CLI, Python, source, documentation, and WASM artifacts so every public surface reports one version.

Out of scope:

- a hosted conversion API, upload endpoint, database, account, or analytics service;
- rendering converted Markdown as active HTML on the playground;
- editing Markdown in the output window;
- OCR, password entry, remote resource fetching, or network-backed filters;
- a service worker or promised offline application in 0.2.0;
- WebAssembly threads, `SharedArrayBuffer`, COOP/COEP headers, or SIMD-gated correctness;
- filesystem-shaped WASI emulation in the browser;
- a Node, npm, TypeScript, React, Vue, Svelte, or other JavaScript build system;
- an npm package publication in 0.2.0;
- replacing Typst as the source of the book or ZDS;
- hand-maintained copies of book chapters, ZDS metadata, capability tables, or benchmark numbers;
- runtime benchmarking of visitors or telemetry about their files;
- claiming quality, coverage, or speed beyond what recorded data establishes.

145. Goals and Design Principles

145.1. Product goals

- Make the first successful browser conversion possible without installation, an account, a tutorial, or an unexplained option.
- Preserve the same artifact, report ordering, manifest, limit, strictness, and detection semantics as the native engine for the same bytes and options.
- Keep documents private by construction: static asset delivery in, no document or result network path out.
- Make the common browser API small, typed, deterministic, and pleasant while retaining the complete result ensemble for advanced callers.
- Keep the page responsive during conversion and make cancellation real by terminating the conversion worker.
- Make errors read like Elm-style explanations and always include a useful next action plus links to the most relevant help.
- Publish `wasm32-freestanding` as a named, downloadable release target with a standalone module and complete browser bundle, not only as a Pages asset.
- Give every supported CLI, Python, WASM, and source target a clear download path with requirements, size, checksum, and provenance.
- Turn the current experimental ZDS index into one coherent project site without breaking published ZDS or PDF URLs.

- Publish the complete book as navigable HTML and PDF from the same Typst sources.
- Give benchmark claims visible method, versions, environment, support matrix, raw data, and reproduction instructions.
- Meet WCAG 2.2 AA for the HTML site and provide usable keyboard, screen-reader, zoom, contrast, reduced-motion, and forced-color behavior.
- Keep repository orchestration under `zig build`; use Python through uv for static-site assembly and validation; use no Ruby.

145.2. System 1: recognition and action

The immediate path is organized around one sentence: **choose a document and get Markdown**. It must provide:

- one visually dominant drop/choose target with an adjacent example action;
- automatic format detection with no format dropdown in the initial path;
- an adjacent output window visibly labeled `Markdown · read only`;
- a persistent privacy statement beside, not beneath, the file action;
- a compact status with ordinary words: loading, ready, converting, complete, canceled, or needs attention;
- copy and download only after output exists;
- a visible `Help & docs` entry in the main header and a contextual help strip directly below the converter;
- no modal onboarding, carousel, surprise popup, or automatic download.

145.3. System 2: inspection and understanding

The deliberate path must expose, without crowding the default path:

- format override, strictness, lowered limits, report details, manifest, and resource inventory behind a clearly named `Advanced options` or `Details` disclosure;
- supported-format and browser-limit reference pages;
- the relevant book chapter from format selection, success, warning, and error states;
- the governing ZDS from the deeper book/help page and diagnostic details;
- benchmark method, raw samples, pinned versions, host, browser, commit, corpus provenance, caveats, and reproduction command;
- source, release artifacts, checksums, security policy, and issue reporting.

145.4. Help is part of the interaction

The book and ZDS are not relegated to the footer. Help follows a three-level ladder:

Level	Question answered	Destination
Quick help	What should I do right now?	Inline hint or concise help page; returns focus to the action that opened it.
Book	How do I use this correctly and what should I expect?	The exact HTML chapter and heading, with a PDF link.
ZDS	Why was the system designed this way and what is the contract?	The exact numbered record and heading, with its PDF and source links.

Every diagnostic code has an optional help mapping generated from the report catalog. A mapped browser error renders `Read how to fix this` beside its directions. General states link to the

quick-start, supported-formats, limits, and security sections. Broken mappings fail `site-check`; the browser never constructs documentation URLs by guessing from a title.

145.5. Original visual language

The visual direction combines bold editorial typography, generous space, a warm paper background, and crisp panels. It is inspired by Notion's confident hierarchy and AnyDoc's direct browser conversion, but it must not copy either site's marks, wording, exact layout, illustrations, or component styling.

The normative design tokens are semantic rather than page-specific:

- display headings use Inter, bundled locally as a Latin-subset variable `woff2`, at 800–900 weight with tight tracking and fluid sizing; body copy uses the same family at 400–500; code uses JetBrains Mono, bundled the same way. Both are licensed under the SIL Open Font License 1.1, and both licence texts ship beside the fonts;
- fonts load with `font-display: swap` and are budgeted separately from the script and stylesheet budget below, at no more than 120 KiB in total;
- the light theme starts with warm off-white paper, near-black ink, soft gray panels, a saturated zen blue action color, and green/amber/red reserved for status;
- the dark theme uses charcoal rather than pure black and preserves the same semantic contrast relationships;
- documentation prose is capped near 72 characters per line; dashboard and workspace panels may be wider;
- corners are modest, shadows rare, borders visible, and icons always paired with text when their meaning is not universal;
- all bundled font licenses ship with the site and release; no font, icon, script, stylesheet, or analytics asset is fetched from a third party.

System is the initial theme. The selector offers `System`, `Light`, and `Dark`. A visitor's explicit choice is stored only in `localStorage`. When there is no stored value, the site follows `prefers-color-scheme`. Native form controls, `color-scheme`, `forced-colors`, print styles, and syntax colors must agree with the active theme.

146. Design Overview

Release 0.2.0 has five connected deliverables:

1. **Pure byte conversion.** Engine entry points needed by the browser accept caller-owned bytes and emit the complete memory artifact ensemble without opening files, starting threads, reading clocks, or consulting environment state.
2. **Zig WebAssembly target.** The root build compiles a no-entry `wasm32-freestanding` executable with an explicit, audited export surface.
3. **Browser adapter and worker.** A small ES module provides the public API; the project site runs it in a dedicated worker and transfers byte buffers.
4. **Static project and documentation site.** Typst produces semantic book/ZDS HTML and PDFs; uv-managed Python assembles stable routes, navigation, search, metadata, fingerprints, and validation; `zig build` owns the graph.
5. **Measured publication.** Native and WASM benchmark records feed one dashboard, and the tag workflow publishes version-matched WASM, CLI, Python, docs, and checksums before Pages deploys the same revision.

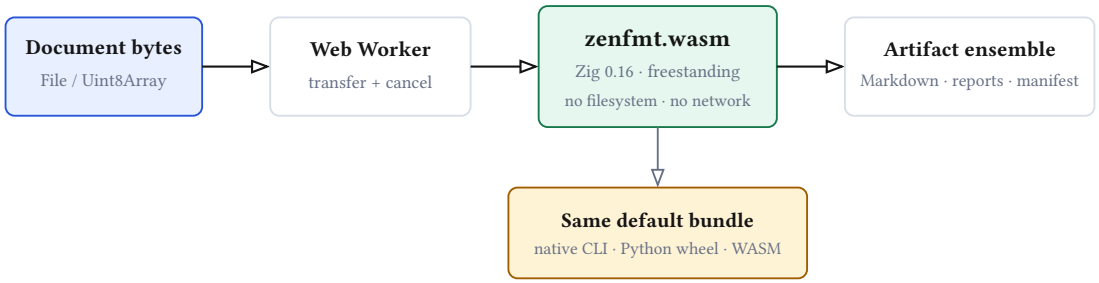

Figure 6: Browser conversion crosses one explicit byte boundary and uses the same default zenfmt bundle as the native and Python releases.

146.1. Deliverable contract

Area	Required 0.2.0 outcome
engine	A filesystem-free, thread-free, deterministic byte conversion path whose artifacts and reports match native memory conversion.
`bindings/wasm/`	Versioned WASM ABI, allocator boundary, capabilities, result accessors, error serialization, import audit, and Zig tests.
browser distribution	ReleaseSafe WASM, ES adapter, worker adapter, declarations, license, readme, and checksum manifest in one versioned archive.
homepage	Responsive local converter, read-only Markdown output window, benchmark dashboard, capabilities, visible privacy, theme selection, and first-class help links to the book and ZDS.
book and ZDS	Multi-page semantic HTML, local search, chapter/record navigation, contextual cross-links, source links, and versioned PDFs.
benchmark	Recorded native and browser lenses with zenfmt, AnyDoc, and Pandoc; correctness gates, support matrix, pinned provenance, raw data, and generated dashboard summaries.
build and CI	Named root build steps, uv locks, browser/accessibility/link tests, preview artifact, protected Pages deployment, and release gates.
release	One 0.2.0 tag and source revision across CLI, Python, WASM, website, book, ZDS, checksums, attestations, and post-publication smoke tests.

147. Zig WebAssembly Implementation

147.1. Target decision

The browser module must be compiled directly by Zig 0.16.0 for `wasm32-freestanding`. Zig's language reference states that it supports WebAssembly out of the box and names the freestanding target for browser hosts. The artifact is an executable with no `_start` entry point; only the versioned ABI functions and linear memory are exported. The wasm32 CPU model is pinned to Zig's `generic` feature set — `bulk_memory`, `multivalue`, `mutable_globals`, `nontrapping_fptoint`, `reference_types`, and `sign_ext` — so the shipped module carries neither SIMD nor atomics and every supported browser can instantiate it.

`ReleaseSafe` is the shipping optimization mode, consistent with the coding standard in ZDS 0002: the released binary keeps its safety checks. The browser is the most hostile input environment zenfmt has, WebAssembly provides no stack guard page, and the engine's bounded validation

frames are sized to `max_depth_hard_cap`; disabling integer-overflow, bounds, and `unreachable` checks there would trade a diagnosable refusal for silent corruption. A `ReleaseSmall` artifact may be produced for size attribution, and may be published in place of `ReleaseSafe` only if `ReleaseSafe` misses the size budget below, which requires a recorded measurement and an amendment.

The module reserves an 8 MiB stack. Because WebAssembly has no guard page, the section auditor asserts that the stack region lies below the first data segment, so a stack overflow traps at low addresses instead of overwriting static data.

WASI is not used for the browser distribution. zenfmt already has a natural byte-in/byte-out boundary, and a filesystem or environment shim would enlarge the authority, bundle, and test surface without improving the public API. The freestanding module must import no filesystem, network, clock, randomness, thread, process, terminal, or host allocation function. The 0.2.0 import allowlist is therefore the empty set: any import at all is an audit failure. A Zig-authored WASM section auditor parses the binary's import, export, memory, global, and custom sections and verifies both allowlists in CI; a textual tool dump is not the security boundary. `Stripped` means the module is built with `strip` enabled and carries no `name` custom section and no `.debug_*` section, which the auditor also verifies.

147.2. 32-bit portability

`usize` is 32 bits on `wasm32`, while every native and Python target zenfmt ships is 64-bit. Size arithmetic that is merely large on a native host can therefore fail to compile, or overflow, in the browser module. Every engine, support, and format library must compile and pass its parity fixtures with a 32-bit `usize`, and byte counts derived from document-controlled dimensions must be computed in a width-explicit type and range-checked before they are used as a length. This is a release gate, not a porting note.

147.3. Engine separation required for WASM

The existing native and Python paths construct `std.Io.Threaded` because they can accept paths and because the Python bridge isolates conversions from an embedding thread's stack. Neither behavior is available or necessary inside a single browser worker. Implementation must separate conversion semantics from host I/O so that:

- browser input is always a named byte slice;
- browser output is always a memory artifact ensemble;
- format readers operate over bounded in-memory readers;
- the selected Markdown writer emits into the existing limited memory sink;
- no path publication, adjacent-manifest lookup, external resource fetch, dynamic plugin loading, or native thread spawn is reachable from the WASM root module;
- capability metadata is generated from the same default bundle definition, not copied into browser code;
- native path and stream modes remain unchanged.

This is factoring, not a browser reimplementation of the engine. Format mapping code, IR construction, lowering, writer output, report catalog, and manifest serialization remain shared Zig modules. Any conditional compilation must sit at host-adapter boundaries and must not fork document semantics.

147.4. Low-level ABI

The WASM ABI is intentionally handle-based. JavaScript sees 32-bit offsets and lengths in linear memory, never Zig struct layout. ABI version 1 provides these operations by role:

Role	Contract
identity	Return ABI version, zenfmt semantic version, build revision, and canonical capability JSON.
allocation	Allocate a 16-byte-aligned caller buffer and free only buffers returned by that allocator.
conversion	Accept request JSON offset/length plus input bytes offset/length; synchronously return an opaque nonzero result handle or the reserved allocation-failure value.
status	Return success, conversion failure, malformed request, or invalid handle, plus the stable exit class.
borrowed result views	Return offset/length pairs for artifact, artifact name, source/output format, report JSON, manifest JSON, and each embedded resource record.
destruction	Free a result exactly once, releasing every arena allocation owned by that conversion.
accounting	Report current and high-water linear-memory pages, live bytes, and live result count, so the host can recycle deliberately rather than guess.

Every offset and length in the ABI is a `u32`. Linear memory on `wasm32` is 32-bit, so a wider type would only force `BigInt` conversions across the boundary for values that cannot exceed `2^32`.

Offset zero means allocation failure and nothing else. A zero-length allocation and every zero-length view instead return a fixed, nonzero, module-owned address with length zero, so a caller never has to distinguish “empty” from “failed” by inspecting a length first.

Result handles are not pointers. The module owns a fixed table of result slots, and a handle carries a generation counter alongside its slot index. A stale, already-freed, or fabricated handle is therefore detected deterministically and answered with the invalid-handle status — never undefined behavior and never a trap. This is what makes double-free and use-after-free defense testable rather than merely asserted.

Result view offsets remain valid until the result is freed. The bytes they address do not move. Host typed-array views over linear memory are a different matter: any call that can grow memory detaches every existing view, so a host must recreate its view after each such call rather than caching one.

The request JSON schema parallels the Python bridge but includes only browser authority: schema, source display name, artifact name, optional `from`, `to`, strictness, `preserve_facets`, and limit overrides. It carries no path and no overwrite flag, and memory is the only output mode, so no output union is exposed. Unknown fields and unknown schema versions are usage errors with structured reports. All lengths are validated before slicing; offset plus length uses checked arithmetic against the current memory size; the module never retains a pointer to request or input memory after conversion returns. A test proves that last property by overwriting the request and input buffers immediately after conversion returns and then reading every result view.

The module exports no C ABI intended for native linking. The WASM ABI and Python ABI may use analogous result concepts but have separate version numbers, because their pointer widths and host lifecycles differ. The build revision reported by the identity role comes from a

build option supplied by the release workflow; absent that option it reads `unknown` rather than fabricating a value.

147.5. Browser profile and memory limits

Browser memory is constrained differently from a native process. The module uses the Zig WebAssembly allocator, wrapped in a counting allocator, with one arena per conversion above it. The wrapper exists because the Zig WebAssembly allocator keeps size-class free lists over grown pages and exposes no accounting of its own; without a wrapper there would be nothing to assert. All temporary and result allocations are owned either by the request buffer or the result handle, and tests prove that alloc/free/convert/free cycles return the wrapper's accounting to baseline even when parsing fails.

Release 0.2.0 defines a browser profile whose defaults are derived from the engine defaults rather than restated, so a limit added to the engine later inherits its engine value here until this record lowers it:

Limit	Browser default	Reason
`max_input_bytes`	32 MiB	The input exists at least twice — the page's copy and the module's — before conversion allocates anything.
`max_total_uncompressed`	128 MiB	Bound archive expansion within the browser worker.
`max_entry_uncompressed`	64 MiB	Prevent one archive member from consuming the entire worker budget.
`max_decoded_text_bytes`	64 MiB	Bound decoded pools independently of compressed bytes.
`max_resource_bytes`	32 MiB	Bound embedded media returned to the page.
`max_output_bytes`	64 MiB	Bound Markdown and other future writer output before another byte is accepted.
`max_nodes`	2,000,000	Keep validation and lowering stacks practical on mobile browsers.
`max_facet_rows`	131,072	At the engine default a facet table can rival the node arrays; the browser bounds both together.
`max_lowering_work`	8,388,608	Bound planner work so a legal but pathological document is refused rather than silently occupying the worker for the full site timeout.

All other engine defaults remain as defined by the release capability data. The public adapter may lower limits. An override **above** a browser-profile value is refused with a structured report rather than silently clamped: the visitor is told the browser profile is the boundary and directed to the CLI or Python library, which is a truthful answer rather than a surprising one. A change to a browser default requires a ZDS amendment and recorded memory measurements.

These per-limit bounds compose, so the record also states the total. Summing the live regions a single worst-case conversion can hold at once — the input copy, the decoded text pool, the node and facet tables, retained resources, the accumulating artifact, and the manifest and report data — and allowing for arena growth and allocator size-class rounding, the worst case is bounded at **1 GiB**. That bound is not advisory: the module is linked with a maximum memory of 1 GiB (16,384 pages), so exhaustion makes memory growth fail, which the allocator reports as out-of-memory, which the engine turns into its canonical structured report. A browser tab is never killed to enforce this limit; the conversion is refused with an explanation.

WebAssembly linear memory cannot shrink. The site therefore recycles its worker at a 512 MiB high-water mark — half the ceiling, so a recycle always precedes a memory refusal — and also after an out-of-memory or trap and after a bounded number of conversions. The public adapter exposes `dispose()`; using a disposed converter is a deterministic usage error. The site discards all references to input, output, reports, manifests, and resources on reset.

147.6. Determinism and artifact parity

For identical input bytes, display name, options, and limits, WASM output must match the native memory API for:

- detected source and selected output format;
- artifact bytes and their BLAKE3-256 digest;
- ordered resource paths, media types, bytes, and digests;
- report severity, code, problem, consequence, context, direction titles, and direction explanations;
- manifest JSON after excluding only the host provenance fields enumerated in the parity test itself, which is the single place that list may live.

Document and resource digests are BLAKE3-256 throughout, matching the engine's manifest algorithm. SHA-256 appears in this record only as the checksum algorithm for published release assets, which is a different concern with a different threat model.

Capability JSON declares target, version, ABI version, supported formats, writer formats, engine limits, the browser profile, the pinned CPU feature set, and unavailable host features. The website renders its format list and advanced controls from this data. A capability disagreement between the site shell and loaded module is fatal and offers reload/download directions; it never guesses.

The browser capability document and the Python bridge's capability document are separate schemas with independent version numbers: the browser document carries a target, an ABI version, and a browser profile that have no meaning for an in-process native bridge. What the two share is their **generators** — the comptime code that walks the default bundle's descriptor tables and the engine limit fields — so neither document can drift from the compiled bundle and neither is hand-maintained.

`zig build capabilities` writes the canonical capability document from the compiled default bundle. The homepage, the input window's accepted-extension list, the download page, and the book's format tables all consume that file; no capability list is authored by hand anywhere in the repository.

148. Public Browser API and Worker Model

148.1. Distribution shape

WASM is a first-class release target named `wasm32-freestanding`, parallel to the native CLI targets rather than an incidental website build output. The release publishes both `zenfmt-0.2.0-wasm32-freestanding.tar.gz`, the complete browser distribution, and `zenfmt-0.2.0-wasm32-freestanding.wasm`, the identical stripped module for low-level consumers. The archive contains:

- `zenfmt.wasm`, the stripped `ReleaseSafe` module;
- `zenfmt.js`, the public standards-based ES module;
- `zenfmt.worker.js`, the optional dedicated-worker adapter used by the site;
- `zenfmt.d.ts`, handwritten declarations checked structurally against the adapter and the API tests;
- `README.md`, `LICENSE`, third-party font/package notices when applicable, and an internal artifact manifest carrying sizes and SHA-256 digests.

The archive has no package-manager assumption. A caller can serve its files from any static origin with `application/wasm` for the module and import the ES adapter.

Release 0.3.5 amends this decision by publishing the same browser distribution as the public npm package `@insanai/zenfmt`. The package adds only metadata and an entry module that exports package relative URLs for the module and worker. It has no dependencies, no generated bundle, and no remote conversion fallback. npm publication consumes the browser archive after its existing audit and conversion smoke test. It does not compile a second module.

This amendment does not change native distribution. The CLI and server remain one self contained executable and do not require Node or npm. The GitHub release archive remains the package manager independent browser distribution. The GitHub Pages site copies these same files from the release build output and adds content hashes to deployed filenames; it does not compile a second WASM variant. CI extracts the archive and proves that its module digest equals the standalone release asset and the module deployed to Pages.

The declarations are validated **structurally**, not type-checked. A TypeScript compiler would require the Node toolchain this record excludes, and buying a type check at the price of a second package ecosystem is a bad trade for one declaration file. Instead a repository-owned checker parses `zenfmt.d.ts`, `zenfmt.js`, and the browser API tests and asserts a three-way agreement: every declared export and member is implemented, every implemented export and member is declared, and both are exercised by a test. It also asserts that every ABI constant appearing in the declarations equals the value the Zig ABI defines. The record claims exactly that much and no more: drift in names and constants is caught, drift in types is not.

148.2. Ergonomic API

The common public sequence is: create a converter from an explicit module URL, await readiness, convert a `File`, `Blob`, `ArrayBuffer`, or `Uint8Array`, read the immutable result, then dispose the converter when its owning scope ends. There is no implicit CDN URL, global singleton, filesystem path, environment lookup, or network conversion fallback.

The API surface has these stable concepts:

Concept	Behavior
`createConverter`	Loads an explicit WASM URL, validates ABI and release versions, exposes capabilities, and resolves only when ready.
`convert`	Accepts browser bytes plus named options and resolves to one complete conversion result; byte-backed inputs require a source name when content detection is insufficient.
`Conversion`	Read-only artifact bytes, UTF-8 `text` only for textual writers, source/output format ids, reports, parsed manifest, resources, elapsed browser timing metadata, and no hidden network state.
`ZenfmtError`	Stable code, exit class, reports, problem, consequence, directions, optional cause, and Elm-style rendered message.
`capabilities`	Frozen snapshot of module identity, formats, limits, and browser-profile restrictions.
`dispose`	Idempotently releases the instance; outstanding conversion behavior is explicit and tested.

Objects and arrays are frozen. Typed-array outputs are defensive copies at the public boundary so later WASM memory growth cannot invalidate or mutate them. The adapter rejects a detached input buffer with an actionable usage error. The worker API accepts an `AbortSignal`; cancellation terminates and replaces the worker, which is the only reliable way to stop synchronous untrusted parsing in 0.2.0.

148.3. State machine

The site owns an explicit state machine:

State	Visible behavior	Allowed transitions
loading	Shell usable; converter says `Loading browser engine…`; file selection may queue one file.	ready or load-failed
ready	Drop target enabled; no result actions.	reading or converting
reading	Selected name/size visible; cancel enabled.	converting, canceled, or failed
converting	Indeterminate progress text; no fabricated percentage.	complete, canceled, timed-out, failed, or trapped
complete	Read-only output, reports, copy/download/reset, and next help.	reading or ready
failed	Elm-style problem and directions; details and help links.	reading or ready

State	Visible behavior	Allowed transitions
load-failed / trapped	Module-specific recovery and CLI/Python fallback.	loading after explicit retry

Only one site conversion runs at a time. Selecting another file during conversion asks for explicit replacement; it never silently discards work. Conversion receives a 30-second site timeout. Timeout terminates the worker and explains that the native CLI permits larger work; it does not claim the document is malformed.

148.4. Elm-style browser diagnostics

All browser-facing failures follow the report contract already required of the engine and Python library. The visible order is:

1. a specific uppercase title;
2. the source display name or operation;
3. what happened in plain language;
4. what was or was not produced;
5. `What you can do:` followed by at least one concrete direction;
6. `Details` for code, exit class, structured context, manifest/report data, and developer cause;
7. contextual `Quick help`, `Read the book`, and when relevant `Read ZDS NNNN` links.

Loader, worker, clipboard, download, browser-support, cancellation, timeout, and ABI failures use the same shape even though they originate outside the engine report catalog. Error snapshots cover light/dark visual states and accessible names. Technical exceptions are never presented alone as `RuntimeError: unreachable`, a numeric WASM trap, or a JavaScript stack.

149. Project Site Information Architecture

149.1. Stable routes

Route	Purpose
`/zenfmt/`	Project homepage, browser converter, benchmark summary, capabilities, quick installation, and primary help paths.
`/zenfmt/benchmark/`	Full native/browser dashboard, methodology, versions, caveats, raw data, and reproduction.
`/zenfmt/book/`	Book start page and chapter tree.
`/zenfmt/book/<chapter>/`	One semantic HTML chapter with local table of contents, previous/next, PDF, related ZDS, and source links.
`/zenfmt/zds/`	Filterable registry of every numbered record.
`/zenfmt/zds/NNNN-<slug>.html`	Stable record URL preserved from the current site.
`/zenfmt/pdf/zenfmt-book.pdf`	Stable latest book PDF; metadata identifies release 0.2.0.
`/zenfmt/pdf/zds-NNNN-<slug>.pdf`	Existing stable per-ZDS PDF URLs.
`/zenfmt/download/`	Current version/date/changelog plus first-class Browser/WASM, macOS, Linux, Windows, Python, and source tar-

Route	Purpose
	gets with requirements, sizes, checksums, attestations, and immutable release links.
`/zenfmt/security/`	Browser privacy model, limits, vulnerability reporting, and support boundary.
`/zenfmt/{zh-hans,ja,ko}/`	Localized converter and project interface for Simplified Chinese, Japanese, and Korean.
`/zenfmt/{zh-hans,ja,ko}/book/`	Localized practical book in semantic HTML, with a matching PDF under `/zenfmt/pdf/`.

All generated URLs are aware of the repository base path and also work under a local root. Internal links are computed as relative paths between routes, so the same output tree serves correctly from a local root and from the repository sub-path with no configuration; no source file hardcodes `/zenfmt/`.

The not-found document is the one exception, and it is a platform constraint rather than a choice. GitHub Pages serves that document for a request of unknown depth, so a relative asset reference in it would resolve against the wrong directory. It alone emits base-absolute references, with the base supplied by the build flag rather than written into a source file.

GitHub Pages also cannot issue an HTTP redirect. A stable alias is therefore either a duplicated file at the alias path or a small document carrying a refresh and a canonical link — never a claimed redirect.

Record URLs keep their existing `.html` form so published links stay valid; new routes are directory-style. The root's former ZDS index moves to `/zds/`, with a clear link from the new homepage rather than an ambiguous redirect loop.

149.2. Language selection and translated books

English is the fallback language. Simplified Chinese, Japanese, and Korean have authored Typst books and localized converter, download, benchmark, and security pages. Each page declares its BCP 47 language in the HTML `lang` attribute. ZDS records remain in English because they preserve the design discussion rather than serving as user documentation.

The first visit to the English converter page may choose a localized homepage from `navigator.languages`. Generic Chinese, Simplified Chinese, China, and Singapore preferences select Simplified Chinese. Traditional Chinese does not silently select Simplified Chinese. Japanese and Korean preferences select their matching routes. An unrecognized preference keeps English.

A visible language selector is present beside search and theme controls. An explicit selection is stored locally and takes precedence over later browser detection. Direct visits to a documentation route are not redirected. English book chapters map to the localized practical book when a reader changes language. ZDS routes remain unchanged when the language preference changes.

Localized books keep commands, API names, report codes, file formats, and established terms such as Markdown, WebAssembly, CLI, API, Pandoc, Docling, and Tika searchable. Surrounding prose uses common, idiomatic language and the same modest claims as the English edition.

Typst does not bundle CJK glyphs. Localized PDF builds therefore name the exact Noto Sans CJK SC, JP, and KR families. CI installs the distribution package that provides those fonts. English book and ZDS builds continue to ignore system fonts and retain their byte reproducibility contract.

149.3. Homepage content order

The default page tells one story in this order:

1. concise identity and value proposition;
2. browser converter with adjacent input and read-only output windows;
3. privacy, local execution, contextual help, and supported-format links;
4. generated benchmark headline and honest dashboard snapshot;
5. format/capability summary and architectural differentiators;
6. a prominent Download entry plus the shortest CLI, Python, and browser-library adoption paths;
7. book and ZDS help cards with descriptions of when each is useful;
8. release, GitHub, security, authorship, and license.

The browser engine loads after the shell becomes interactive. Book and ZDS navigation must remain usable when WebAssembly, workers, JavaScript, or file APIs are unavailable. A load failure changes only the converter panel and shows CLI/Python alternatives; it does not turn the project site into an error page.

The homepage also contains a compact `Download zenfmt` section. It presents Browser/WASM, macOS, Linux, Windows, and Python as labeled choices, highlights the likely platform only when detection is reliable, and links to the full download page for architecture/libc choices, checksums, source, and previous releases. The compact section never replaces exact target selection with a single opaque `Download` button.

149.4. Download experience

`Download` is a persistent global-navigation item and a visible homepage call to action. The dedicated page borrows the useful information hierarchy of Zed's download page—current version and date first, changelog next, then clear platform choices—without copying Zed's visual components or wording. zenfmt has no preview channel in 0.2.0, so the page shows `Stable 0.2.0` and does not display a disabled or fictional channel switcher.

The top of the page contains release version, release date, release notes, source revision, checksums, attestations, and browser/security notes. The Browser/WebAssembly target is a full-width first-class card because it is the new 0.2.0 surface. Native CLI, Python, and source targets follow in a stable grid:

Family	Target choices	Primary action and guidance
Browser / WASM	`wasm32-freestanding`	Download the complete browser archive; secondary links expose the standalone `.wasm`, adapter files, declarations, README, and integration chapter.
macOS CLI	Apple Silicon, Intel	One direct ReleaseSafe archive per architecture, minimum macOS version, size, SHA-256, and installation steps.

Family	Target choices	Primary action and guidance
Linux CLI	x86_64 or ARM64; glibc 2.17+ or musl	A labeled target chooser explains libc and architecture before download; no unlabeled filename puzzle.
Windows CLI	Intel/AMD 64-bit	Portable `.zip`, system requirement, size, SHA-256, extraction, and PATH guidance.
Python library	CPython 3.10–3.14 on supported wheel platforms	Show `uv add zenfmt` as the common action, link PyPI, and expose all wheel and source-distribution files for auditing.
Source	tagged repository archive	Download source, verify revision, or follow the Zig build chapter.

Client hints may highlight a likely native platform as `Recommended for this device` only when platform and architecture are reliable. Recommendation never hides, reorders, or automatically starts another target; all choices stay one click away. If architecture or libc is uncertain, the page asks the user to choose and gives a one-sentence explanation. User-agent guessing must not select a binary silently.

Every primary action names the artifact, version, target, archive type, download size, and minimum runtime requirement before activation. SHA-256 and attestation links are beside the artifact rather than buried in release notes. All URLs are generated from the 0.2.0 release manifest and point to immutable, tag-specific GitHub release assets or the immutable PyPI version page. A missing asset, size/digest disagreement, unsupported target label, or link to `latest` fails `site-check`.

149.5. Download page wireframe

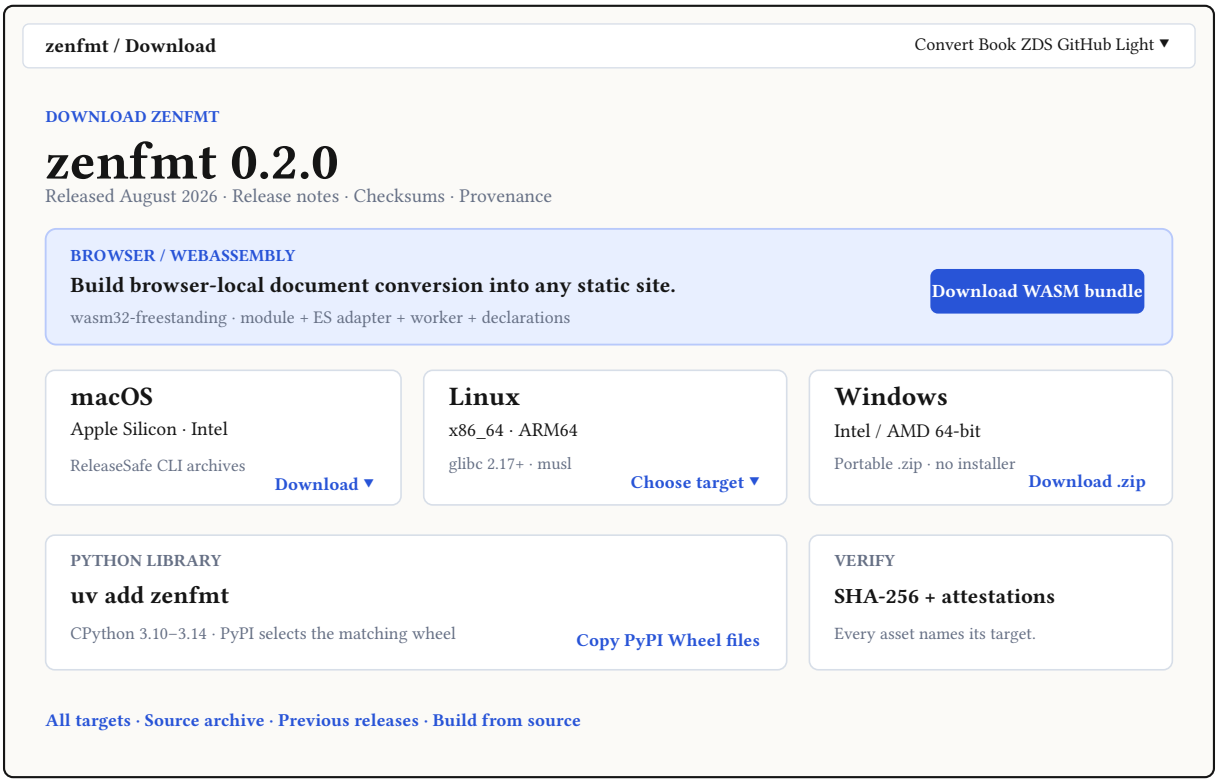

Figure 7: Download page. The current release and first-class WASM target lead; native and Python choices remain scannable, explicit, and verifiable.

149.6. Desktop homepage wireframe

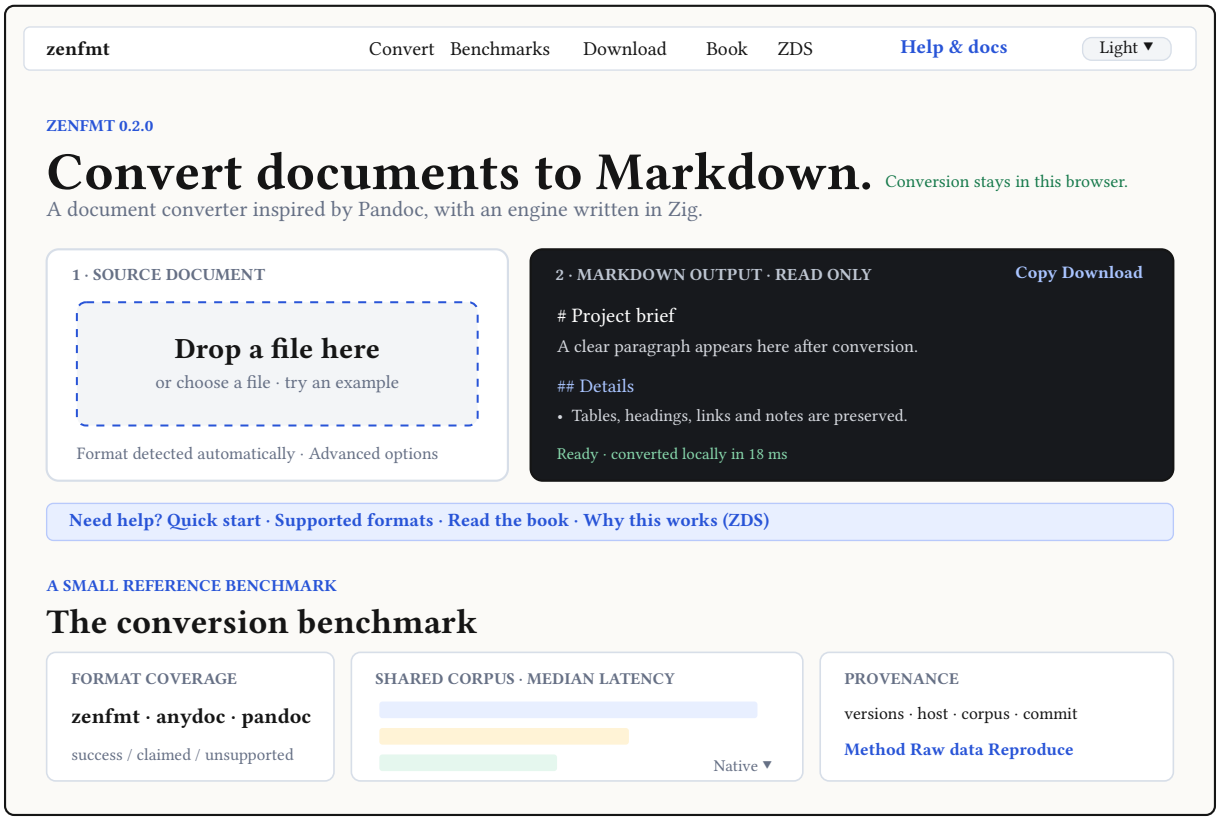

Figure 8: Desktop homepage. The adjacent dark panel is a wider visual “window” for readable Markdown, not a browser popup. Help and benchmark evidence sit inside the primary reading path.

149.7. Documentation wireframe

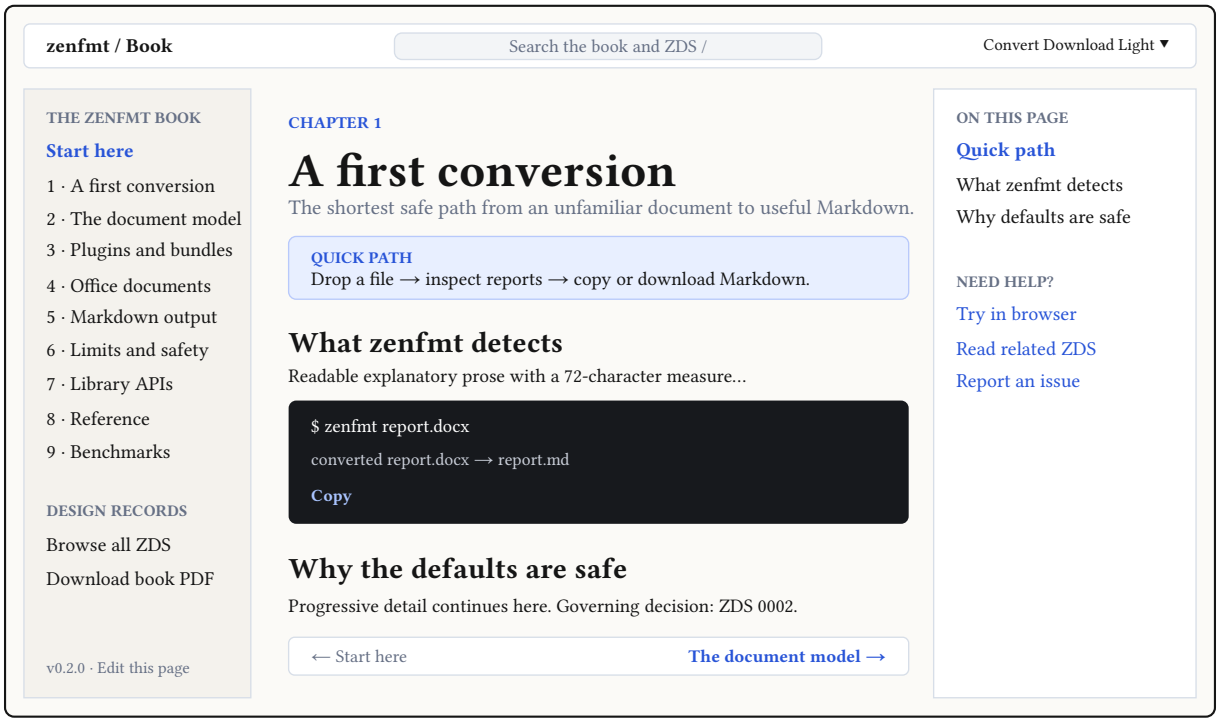

Figure 9: Book/ZDS documentation shell: global search, stable left tree, focused article, local table of contents, and contextual help.

149.8. Mobile wireframe

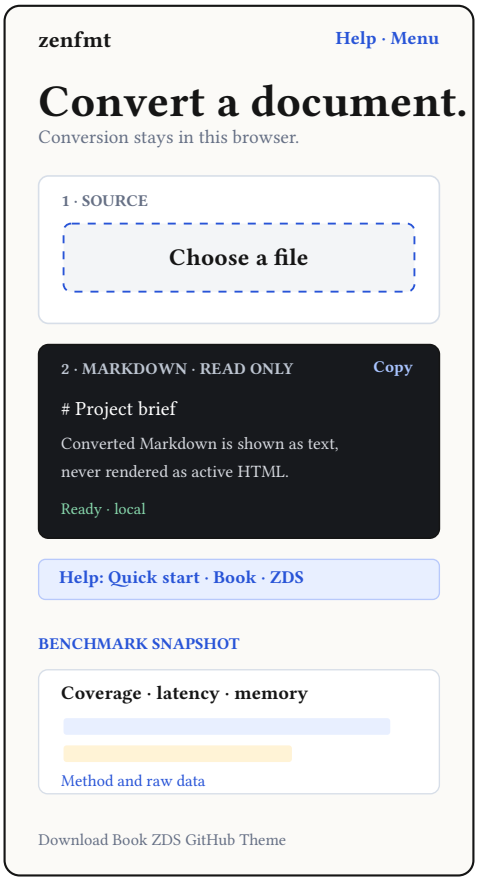

Figure 10: Mobile preserves the same task order by stacking input, output, help, and benchmark. No essential action depends on drag-and-drop.

150. Converter Interaction Specification

150.1. Input window

The entire labeled drop area is one keyboard-focusable file action backed by a native file input. It accepts drag/drop, click/tap, keyboard activation, and an included safe example. Drag is never the only way to choose a file. The input window shows accepted extensions from capability data but says that content detection is authoritative. It displays name and human-readable size before conversion and asks before replacing a running selection.

The browser reads one file only. Directory upload, multiple conversion, paste of filesystem paths, and remote URLs are absent in 0.2.0. Byte contents are not logged. The document name is used only for detection fallback, reports, manifest provenance, and suggested output filename.

Advanced options are closed initially. When opened they expose source format override, target writer, strictness, facet preservation, and browser-safe limit reductions. Each option has a one-sentence inline explanation and a book link. Changing an option after a result exists marks that result stale and offers an explicit reconvert action.

150.2. Output window

The second window is adjacent on genuinely wide screens and stacked immediately below input everywhere else. The workspace may grow to 1600 CSS pixels. In side-by-side mode the input uses approximately 40% and Markdown approximately 60%; the output's inner content measure must provide at least 80 monospace characters after padding and is capped near 100 characters

so prose does not become a hard-to-track ribbon. The layout must stack rather than squeeze the output below that minimum.

The output uses a semantic `pre`/`code` region with `tabindex=0`, an accessible label, selectable text, and no editing behavior. It defaults to soft line wrapping for readable prose and provides a labeled `Wrap lines` toggle for tables, code, or callers who need exact visual line boundaries. Wrapped and unwrapped modes preserve the same Markdown bytes; unwrapped overflow scrolls inside the code window, never the page. The reading viewport is at least 24rem high after conversion and may grow to 70% of the viewport before it scrolls. Content is inserted through text nodes/ `textContent` only. It is never passed through a Markdown renderer and never assigned to `innerHTML`.

The toolbar contains Copy, Download Markdown, Details, and Reset. Copy requires an explicit gesture and confirms success without moving focus. Download uses a sanitized deterministic filename derived from the artifact name. Embedded resources appear in Details with independent download actions; the page does not invent an archive format. Warnings remain visible after success and link to their explanations.

150.3. Progress, status, and focus

File reading may report byte progress when the browser supplies it. Native conversion is indeterminate in 0.2.0 and uses honest status text, not a fake percentage or animated timeline. `prefers-reduced-motion` removes nonessential transitions. Completion does not steal focus; an `aria-live=polite` region announces status and the user can move to output with one explicit shortcut. Errors receive programmatic focus only after the initiating action and return focus to the file control after reset.

150.4. Responsive behavior

- Side-by-side mode begins around 1280 CSS pixels only when a container query confirms that the output retains at least an 80-character readable measure; zoom, font metrics, and neighboring controls may force stacking sooner.
- In side-by-side mode input/output use approximately 40/60 width with a 24-pixel minimum gutter. In stacked mode output follows input in DOM and visual order and receives the full workspace width.
- Navigation collapses to a labeled menu; `Help` remains directly visible.
- Documentation's left tree becomes a drawer, the right table of contents becomes an in-article disclosure, and search remains in the header.
- At 200% zoom and 320 CSS-pixel width, no action or prose requires horizontal page scrolling; only code/data tables may scroll within labeled regions.

151. Book and ZDS as HTML Documentation

151.1. One source, two reading modes

Typst remains the canonical source. PDF is the archival/print edition; HTML is the navigable, accessible web edition. No generator converts PDF back into HTML and no chapter is manually duplicated.

The implementation adds a book bundle entry point that emits one HTML document per chapter. The archival PDF continues to come from its own separate Typst invocation. The two must not share one compilation: a bundle that emits the same content as both HTML and PDF compiles that content twice in a single pass, which collides the labels the book relies on for its outline, figure references, and cross-chapter links. The ZDS records already live with that constraint by

cross-referencing in prose; the book, which cannot, gets two invocations instead. For the same reason a reference that crosses chapters is resolved through the checked-in content map rather than a Typst reference: in bundle mode such a reference resolves and then emits a fragment pointing into a different output file, which the exporter cannot express.

Existing chapter files gain target-aware presentation only where paged layout cannot map semantically to the web. Equations use semantic MathML where Typst supports it. Tables retain headers and captions.

Figures need more than a caption, because of how Typst renders them. A diagram exported to HTML becomes vector artwork in which **every character of text is a glyph outline** — there is no text content in the output at all. Such a figure is unreadable to a screen reader, unsearchable, and unselectable no matter how good the drawing is. Therefore every figure that communicates structure declares a text alternative, and the complex ones additionally declare a long description that conveys what the diagram shows in prose; a figure that declares neither, and is not marked decorative, fails the build. Decorative visuals are hidden from assistive technology.

The same rendering has a size consequence. Inline diagram artwork can exceed the surrounding prose by an order of magnitude, so the assembler extracts each one into a separate fingerprinted image file, references it with its declared alternative text, and loads it lazily. This keeps documentation pages within the page-weight budget, lets diagrams cache independently, and removes another source of inline styling.

Typst 0.15's HTML and bundle export is still documented as experimental, and its failure mode is silent: constructs it cannot map are dropped without an error naming them. The release therefore pins the exact Typst patch version, snapshots the semantic DOM for representative pages, and treats an upgrade as a reviewed toolchain change. The Python assembler wraps and indexes Typst output but does not repair unknown semantic breakage silently. It enforces that promise with an explicit contract check before wrapping: an element or attribute outside the allowed set, a style value outside the closed map, a figure count that disagrees with the content map, an empty figure body, a missing figure alternative, or a Typst warning outside the reviewed allowlist each fail the build with the offending fragment quoted.

Typst identifies headings by their position in the document, so those identifiers renumber whenever any earlier content changes and cannot serve as stable anchors. The assembler generates anchors by slugging heading text within each chapter or record, rewrites every internal fragment to match, and retains the previous identifier as a secondary identifier for one release so existing links keep working.

151.2. Sphinx-quality documentation ergonomics

Every book chapter and ZDS page provides:

- a skip link and semantic landmarks;
- project header with Convert, Download, Book, ZDS, Benchmark, GitHub, search, and theme;
- stable left navigation with the active location and collapsible groups;
- an article heading hierarchy with unique slugged anchors and visible permalink controls;
- an `On this page` table of contents on wide screens;
- previous/next navigation where order exists;
- `Edit this page`, source revision, release version, HTML/PDF, and related records;
- copy buttons for copyable code without blocking ordinary text selection;
- a local, keyboard-operated search dialog that works without a network;
- print CSS that removes navigation without hiding content.

The book's editorial structure follows the same dual path as the product. Each chapter begins with a short `Quick path` or learning objective, then develops the mental model, details, failure modes, and governing decisions. Reference tables remain scan-friendly. ZDS pages prioritize rationale, contracts, alternatives, and acceptance gates; they do not masquerade as tutorials.

151.3. Navigation and contextual linking

A small checked-in content map identifies:

- book chapter order and route;
- ZDS registry entries and route;
- format id to book section and format ZDS;
- report code prefix to quick help/book heading/ZDS heading;
- API surface to book reference heading;
- benchmark panel to method heading and raw result file.

The site generator validates every destination and emits the map for the browser. Links use descriptive text such as `Understand archive limits`, not `Learn more`. Each book page can show `Related design records`; each ZDS can show `Read the user-facing chapter`. The homepage has visible Book and ZDS links in the header, contextual help strip, dedicated help section, and footer.

151.4. Search

The uv-managed Python generator builds a compact JSON index from final semantic HTML: page title, hierarchy breadcrumbs, heading, concise visible text, route, and content kind. It excludes navigation, code-generated noise, benchmark raw samples, and hidden text. The browser search module performs deterministic token/prefix matching locally and groups results by Book, ZDS, Reference, and Site. Search has no remote endpoint, artificial intelligence claim, tracking, or document access.

The `/` shortcut focuses search only when focus is not in a control. Escape closes it and restores focus. Results expose matched context and the target section. The no-JavaScript fallback remains the book/ZDS navigation tree and generated indexes.

151.5. PDF publication

`zig build book` continues to produce the complete PDF. Release 0.2.0 adds version metadata, outline/bookmarks, document language, author names without email, descriptive title, link annotations, alt text where Typst supports it, and a reproducible creation policy. The build targets PDF/UA-1 when the pinned Typst toolchain and templates pass validation; otherwise the site must not claim PDF/UA conformance and HTML remains the primary accessibility target. Text extraction, bookmarks, links, page order, metadata, and representative screen-reader reading order are still release gates.

Each ZDS PDF retains its stable URL. The homepage, help menu, book header, ZDS index, and download page link the book PDF. Each HTML ZDS links its own PDF and source. The ZDS PDFs are published through GitHub Pages and are not duplicated as GitHub Release assets. PDF download size is visible before activation.

152. Benchmark Dashboard

152.1. What the front page may claim

The homepage may show only generated facts from result files belonging to the same release and benchmarked implementation revision. A following data-only commit may add those raw/

generated results and this record's lifecycle state; the release gate proves no conversion, binding, build, or site source changed after the recorded revision. The dashboard must never contain a typed-in latency, ratio, coverage count, bundle size, or ranking. Public copy describes the corpus and observed ratios without declaring a winner. Comparisons use the geometric mean of per-file ratios over files both tools successfully convert. Unsupported and failed files remain visible and are never treated as infinitely slow.

The dashboard presents three separate questions:

- **Coverage**: which corpus formats and files each tool claims and successfully converts;
- **Performance**: cold startup, warm conversion latency, CPU or browser time, peak native RSS or WASM high-water memory, and download size where meaningful;
- **Output preservation**: deterministic structure and text checks against a tool-neutral fixture oracle, reported by format and never collapsed into an unsupported "universal quality" claim.

152.2. Native lens

The existing `zig build benchmark` remains the native lens and compares the released zenfmt CLI, clean-installed zenfmt wheel, pinned Pandoc, and pinned AnyDoc. It measures median wall latency, CPU time, and peak RSS after one discarded warm-up, with process startup included for CLI rows. Warm Python API results remain a separate profile as defined by ZDS 0014. The site must label these semantics next to the selector and link the full method.

152.3. Browser lens

A new pinned browser harness measures release artifacts for zenfmt WASM, AnyDoc WASM, and a documented Pandoc WASM distribution when each can perform the same file-to-Markdown task. Competitor artifacts are fetched only by the reproduction/setup step, pinned by version and digest, and never silently updated. Their licenses and invocation adapters are recorded. They need not be shipped to site visitors.

Browser measurements run in a fresh profile with cache state declared:

- cold fetch, compile, instantiate, and first conversion are separate fields;
- warm conversion uses three warm-ups and at least fifteen measured iterations per file, retaining raw samples, median, p95, and median absolute deviation;
- input and result transfer time is reported separately from WASM execution where the host API permits it;
- WASM high-water pages and JavaScript heap are recorded only when the browser exposes stable measurement APIs; missing metrics are `unavailable`, never zero;
- raw and compressed artifact sizes are taken from the released files;
- the browser name/version, OS, architecture, CPU, memory, power mode, runner, commit, artifact digests, and date are stored with results;
- each tool uses its documented browser API and its release optimization mode.

Pandoc and AnyDoc may support different subsets under WASM. The support matrix is the first chart. Head-to-head speed exists only for shared successful files. If no maintained, reproducible Pandoc WASM artifact meets the harness contract, the dashboard shows Pandoc as `not benchmarked in browser` with the reason and retains Pandoc in the native lens; it does not substitute native Pandoc timing into a WASM chart.

152.4. Correctness before timing

Every timed output first passes:

- nonempty and valid UTF-8 Markdown when success is claimed;

- normalized text-preservation checks against fixture truth;
- structural probes for headings, lists, tables, links, code, notes, and sheet or slide boundaries that exist in the source fixture;
- stable artifact digest across repeated same-tool conversions;
- zenfmt WASM/native parity checks defined earlier;
- explicit exclusion with a recorded reason for any failed correctness gate.

The fixture oracle records document facts rather than zenfmt's preferred Markdown spelling. Quality scores are reported per feature and format with the scorable document count. Adding an LLM judge is outside 0.2.0 unless a later record defines model/version, images, prompts, blindness, position-bias control, cost, raw verdict publication, and rerun policy.

152.5. Result schema and generated presentation

`benchmarks/results/site.json` is the dashboard source. It references native `latest.json`, Python `python.json`, browser `wasm.json`, quality results, corpus manifest, environment, and tool manifests by SHA-256 digest. Schema version, zenfmt version, and git revision are mandatory. Python generation calculates aggregates from raw samples; the Typst book benchmark chapter and HTML dashboard consume those aggregates from the same file and recompute nothing. `site.json` is checked in, so a documentation build from a clean checkout needs no benchmark run.

Raw data means raw timing samples and per-file measurements. The corpus is assembled from third-party documents whose licences differ and are in several cases unstated, and a conversion of such a document is a derivative work. Neither the corpus documents nor their conversions are published. The corpus manifest publishes each file's identifier, declared format, byte size, SHA-256 digest, source URL, and licence, which is what a reader needs to reproduce the run; the fetch script verifies those digests and fails rather than silently benchmarking a file that changed underneath it.

The homepage shows a small dashboard: coverage, one shared-corpus latency comparison for the selected lens, and provenance/reproduction links. The full page adds support matrix, per-file bars, cold/warm split, memory, bundle size, quality probes, raw table, method, caveats, and accessible data tables behind each chart. Color is never the only carrier of comparison, failure, or unsupported state. Tool order is stable: zenfmt, AnyDoc, Pandoc.

Reference results are regenerated deliberately on the designated host for a release candidate. Ordinary Pages builds validate but do not rewrite them. Results whose version or revision does not match the build make the dashboard fail closed with `Benchmark pending for this release`, everywhere including the homepage headline; stale 0.1.0 numbers cannot appear under a 0.2.0 heading.

That mismatch is a hard failure on the default branch and on a tag, where the reference data is expected to match the release candidate's revision. On any other branch it is a recorded warning, because otherwise every commit that did not regenerate the benchmark would block a documentation change on a benchmark run — a gate that would be disabled within a week rather than obeyed.

153. Security and Privacy

153.1. Threat model

Inputs are hostile documents chosen by arbitrary visitors. Attack goals include archive bombs, parser bugs, excessive allocation, long-running work, malicious filenames, script injection

through Markdown/reports/metadata, resource URL exfiltration, worker escape, supply-chain substitution, and stale or misleading benchmark data. GitHub Pages and release assets are public; there are no secrets in the site build or browser runtime.

153.2. Capability minimization

The WASM import audit proves the release module has no network, filesystem, clock, randomness, process, terminal, or thread imports. The browser adapter fetches only the explicit same-origin module URL. Conversion never follows an external document relationship, URL, adjacent manifest, include, font, image, or stylesheet. External references remain inert manifest/report text.

The site has no analytics, advertising, session replay, remote font, tag manager, user account, upload endpoint, or conversion fallback. Theme choice is the only persisted setting. It stores no document name, bytes, output, reports, recent-file list, or benchmark behavior.

153.3. Content handling

- Untrusted Markdown, report text, filenames, manifest values, and resource names enter the DOM only as text.
- Filenames are stripped of path components, control characters, bidi control surprises, and unsafe download characters before display or download.
- Blob URLs are created only for explicit downloads and revoked promptly.
- Clipboard write happens only after the visitor presses Copy.
- Error causes and stacks are available in developer details without leaking unrelated browser state.
- Example fixtures are authored, licensed, small, and served same-origin.

153.4. Browser isolation and policy

Conversion runs in a dedicated worker. GitHub Pages serves static files and gives no control over response headers, so the policy is a document policy carried in a meta element. That is a real constraint, and this record states what it costs rather than implying protection the platform cannot provide.

A document policy enforces the fetch directives: default, script, style, worker, connect, image, font, object, base, and form. The site sets all of them, and `object-src 'none'`, no `eval`, no inline event handlers, and no third-party origin. It also sets `require-trusted-types-for 'script'` with an empty `trusted-types` list, which a document policy does honor where supported and which turns an accidental `innerHTML` assignment — the one mistake that would undo the read-only output guarantee — into a runtime failure rather than an injection.

A document policy does **not** enforce `frame-ancestors`, `sandbox`, or violation reporting, and GitHub Pages sets no framing header of its own. Release 0.2.0 therefore has **no clickjacking protection**: the site can be embedded in a third-party frame. That is recorded here as an accepted gap and stated on the security page. It is tolerable only because the site holds no credential, no session, and no server-side action a framing attacker could induce; it would not be tolerable for a site that did. Response headers, and with them framing protection and violation reporting, require a hosting change and a later record.

WASM execution receives one narrow allowance: `script-src` carries `'wasm-unsafe-eval'` alongside `'self'`, which Chromium requires to instantiate a module under a restrictive policy and which the other supported engines ignore. No other `eval`-class token appears.

Typst's HTML export emits a stylesheet element and a large number of inline style attributes, chiefly for syntax highlighting. A policy cannot cover style attributes by hash without weakening it further, so the assembler removes them instead: it extracts every emitted stylesheet element into a fingerprinted external stylesheet and rewrites every inline style attribute through a closed map of semantic classes. A style value not in that map fails the build rather than being allowed through. The published site therefore contains no style element and no style attribute at all, and `style-src 'self'` holds with no exception.

No feature requires cross-origin isolation. WASM threads and `SharedArrayBuffer` are excluded, which avoids dependence on COOP/COEP headers. A generated policy test inventories every deployed URL, asserts the policy text is identical on every page, and fails on an undeclared external request, on any style element or style attribute, and on any inline event handler.

153.5. Resource exhaustion and recovery

Browser-profile limits fail in the engine before the next bounded resource is accepted. Worker timeout handles computation that is legal but too slow for the site. Traps, OOM, and timeout terminate the worker, invalidate its views, free browser references, and create a clean instance before retry. The visible error distinguishes document refusal, browser profile limit, timeout, trap, and module load failure, each with specific CLI/Python fallback directions.

Native fuzz targets remain authoritative for format parsers; the WASM boundary adds malformed request, offset/length, allocation/free ordering, and repeated conversion fuzz/property tests. A representative adversarial corpus runs in real browsers under time and memory budgets.

154. Accessibility and Human Interface Gates

The HTML target is WCAG 2.2 AA. Automated checks are necessary but not sufficient. Release review includes keyboard-only, screen-reader, zoom, forced-colors, reduced-motion, touch, and high-contrast manual passes.

Required properties include:

- semantic header, navigation, main, article, aside, footer, headings, lists, tables, buttons, labels, status, alerts, and code regions;
- visible skip link and focus ring with focus never obscured by sticky UI;
- 44-by-44 CSS-pixel preferred pointer targets and the WCAG minimum where layout cannot provide that size;
- text contrast at least 4.5:1, large text 3:1, meaningful UI graphics and focus indicators 3:1 in light and dark themes;
- no information conveyed by color, motion, position, or icon alone;
- drag/drop alternatives, labeled file input, and no keyboard trap in code, search, nav drawer, disclosure, or dialog;
- status announcements that do not repeat on every render;
- logical DOM order matching visual order at all breakpoints;
- charts paired with data tables and prose summaries;
- locale-independent machine data, authored English, Simplified Chinese, Japanese, and Korean human copy, with language declared;
- 200% text zoom and 400% page zoom checks without loss of actions or content.

Usability acceptance uses five first-run tasks with no verbal coaching: find a file, recognize local-only processing, copy output, recover from an unsupported or oversized input, and reach the relevant book/ZDS help. Every task must have an obvious first action, visible system status, and

a successful keyboard path. Findings and fixes are recorded with the release candidate; this is an engineering test, not collection of visitor telemetry.

155. Build and Repository Integration

155.1. Proposed repository shape

The implementation adds these responsibilities without adopting a JavaScript project manager:

```
bindings/shared/           comptime capability and name generators
bindings/wasm/             Zig ABI, request/result ownership, browser profile
build/                     the split build graph: modules, wasm, python, zds
site/                      authored shell assets and homepage content
  assets/                  CSS, ES modules, worker, local fonts, icons
  templates/               the shared HTML shell and page templates
  pages/                   body partials and microcopy for generated pages
docs/book/site.typ         multi-document Typst book bundle entry point
docs/i18n/                 Simplified Chinese, Japanese, and Korean Typst books
docs/site/                 uv-managed Python assembler and validators
tools/wasm_audit.zig       WASM section auditor
tests/wasm/                Zig parity, ABI, limit, leak, and adversarial tests
tests/site/                Python and browser interaction/accessibility tests
benchmarks/browser/        browser adapters and uv-managed benchmark runner
benchmarks/results/        native, Python, WASM, quality, and site JSON
```

This tree is an implementation map, not code added by this ZDS.

The pages outside the book and the ZDS records — the homepage, download, benchmark, security, and not-found routes — are authored as HTML templates rendered by the Python assembler, not as Typst. Every one of them is data-driven: it must inject the release manifest, benchmark aggregates, and capability metadata, and it must express file inputs, live regions, disclosures, and a no-script fallback. Typst's HTML export is experimental, cannot emit into the document head, and drops constructs it cannot map without failing; the product surface should not depend on it. Typst therefore retains authority over exactly what it is the canonical source for: the book and the records.

155.2. Division of tool authority

- Zig 0.16.0 compiles native and WASM artifacts, runs Zig tests/audits, and owns the repository build DAG.
- Typst 0.15.1 — pinned to that exact patch version — produces semantic bundle HTML and PDFs from the book/ZDS sources. Its `@preview` imports name exact package versions and the Typst CLI resolves and caches them itself; the sources are not vendored, because an exact version in the import already pins what is resolved, and copying third-party packages into this repository would mean carrying their licences inside an MIT project for no additional guarantee.
- uv manages locked Python environments for site assembly, indexing, validation, benchmark aggregation, and browser tests. The site tooling is its own uv project with its own lockfile, kept separate from the published Python library so its development dependencies never enter that distribution.
- Ruff formats/lints all new Python tooling; pytest tests it.
- Small browser ES modules and CSS are checked-in runtime assets. They use web standards directly and require no transpiler, bundler, or package-manager lock.
- Node and npm are not a JavaScript build system or a shipped runtime dependency. The repository contains one authored `package.json` under `packages/wasm` as publication metadata. There is no lockfile, `node_modules`, transpilation, bundling, or generation step. The release

workflow uses npm only to inspect and publish the already tested browser files. Test tools distributed as locked Python wheels may embed their own private runtimes, and npm may fetch a pinned benchmark competitor into a gitignored directory.

- Ruby is not part of authoring, generation, testing, serving, benchmarking, release, or deployment.

155.3. Root build steps

Step	Required behavior
`zig build capabilities`	Write the canonical capability document from the compiled default bundle; every other surface consumes it.
`zig build wasm`	Build the host-independent ReleaseSafe module and browser distribution into `zig-out/wasm`.
`zig build wasm-check`	Run the ABI/import/export audit, the freestanding WASM tests, native parity fixtures, leak cycles, declaration checks, and adapter contract tests.
`zig build book-site`	Build chapter HTML from the book sources; the versioned book PDF stays on its own invocation so the two never share one Typst compilation.
`zig build zds-site`	Build registry, per-ZDS HTML, and PDFs under the shared site contract; existing command name remains valid.
`zig build site`	Build WASM, book, ZDS, homepage, download and benchmark pages, search, fingerprints, aliases, `.nojekyll`, and the final deploy directory.
`zig build site-check`	Validate HTML semantics, internal/external links, base paths, CSP inventory, search map, version/digest parity, performance budgets, and browser tests.
`zig build site-serve`	Serve the final directory through uv-managed Python on localhost with the correct WASM MIME type; no production server.
`zig build benchmark-browser`	Run pinned browser comparison and write raw `wasm.json`; never run automatically for visitors.
`zig build benchmark-quality`	Check every tool's output against the fixture oracle and write the quality result file the dashboard requires.
`zig build benchmark-aggregate`	Compute dashboard aggregates once from the raw result files and write `site.json`.
`zig build docs`	Build book/ZDS PDF and HTML plus site documentation; it remains usable without building release artifacts unrelated to docs.

`zig build test` depends on deterministic unit/parity tests but not the long reference benchmark, and not on any WebAssembly compilation: the browser bundle is a host-independent conversion bundle, so its parity, limit, and reachability tests run natively and cheaply. `zig build fmt-check`

includes Zig formatting and Ruff format/lint checks for site Python. `site-check` builds into a clean directory twice where reproducibility is asserted and rejects source-tree leakage.

Reproducibility across those two builds is not automatic and depends on mechanisms this record requires rather than hopes for. Every Typst invocation passes an explicit creation timestamp drawn from `SOURCE_DATE_EPOCH` and ignores system fonts, using only the fonts Typst itself embeds. Ignoring system fonts is not a nicety: Typst's HTML figure export embeds glyph outlines, so a font that resolves differently on another machine changes the generated HTML byte-for-byte and not only the PDF. A document must therefore name only fonts Typst embeds, and `site-check` is what catches one that does not. Generated archives are written with fixed modification times, sorted entries, and zeroed ownership.

155.4. Static assembly

The Python assembler receives only explicit build inputs and an output directory. It generates navigation, breadcrumbs, search index, help map, benchmark summaries, asset fingerprints, metadata, feeds/sitemap if included, and stable aliases. It copies no arbitrary repository file. Output paths are normalized, confined below the destination, and collision checked.

The final Pages directory includes `.nojekyll`, a useful `404.html`, the fingerprinted browser assets, stable HTML routes, book and ZDS PDFs, benchmark JSON, and license notices. It contains no source map exposing local paths, uv environment, cache, corpus document, temporary output, unpublished ZDS draft, or GitHub token.

156. Testing and Quality Gates

156.1. WASM tests

- Compile the complete default bundle for `wasm32-freestanding` in CI.
- Validate exact import/export lists and ABI/version/capability schema.
- Exercise every reader family through the low-level ABI in a real WASM runtime and through the public browser adapter.
- Compare every committed parity fixture with native memory conversion.
- Test allocation failure, invalid handles, overflowed offset/length, malformed JSON/UTF-8, double-dispose defense, empty input, large input refusal, traps, worker timeout, cancellation, and worker recycle.
- Run repeated success/failure cycles and assert bounded allocator/high-water behavior.
- Test `File`, `Blob`, `ArrayBuffer`, `Uint8Array`, detached buffer, missing source name, explicit format, strictness, lowered limits, resources, warnings, and Elm-style errors.

Most of that runs natively and costs milliseconds, because the browser bundle differs from the native bundle only in host authority, not in target: it is a conversion bundle with the filesystem compiled out, so parity, limits, and reachability are ordinary Zig tests. What genuinely needs a runtime is narrow — that the compiled module instantiates, that the reserved stack survives a maximally nested document, and that the adapter behaves in a browser.

Zig's default test runner cannot be compiled for `wasm32-freestanding`: it requires process arguments, threaded I/O, and standard input. The freestanding tests therefore run through a small repository-owned test runner that exports a result function and a failure log, instantiated by a same-origin harness page under browser automation.

156.2. Browser coverage

The supported-browser statement below describes what the site is built to work in, determined by capability detection. It is not a claim about what is continuously tested, and this record keeps the two separate.

Release 0.2.0 gates on Chromium. The interaction, adapter, accessibility, and harness suites run there on every change, and a failure blocks the release. Firefox and WebKit are exercised on a best-effort basis and their failures are recorded rather than blocking, because standing up and stabilising three engines is more work than this release can absorb without displacing the correctness gates that matter more. That is a deliberate reduction from the original intent and is recorded as such; widening the gate is a later change, not a silent one.

Automated WebKit coverage, when it arrives, is also not Safari: the automation stack ships its own WebKit build, which differs from the shipping browser in areas — WebAssembly tiering, worker lifecycle, policy enforcement — that are exactly where a converter defect would hide. Release acceptance therefore includes one manual pass on current stable Safari on macOS and one on iOS, and the site never claims tested support it does not have.

156.3. Site and documentation tests

- Snapshot representative homepage, book, ZDS, benchmark, error, empty, loading, success, dark, and high-contrast states at desktop and mobile widths.
- Test keyboard order, skip link, drop alternative, theme control, search, language selection, browser-language default, disclosures, copy, download, cancel, reset, nav drawer, help ladder, and focus restoration.
- Run automated WCAG checks in all three theme selections and manually review the tasks listed earlier.
- Validate unique headings/ids, landmarks, labels, alt text, table headers, BCP 47 language, canonical metadata, Open Graph text, and no empty links.
- Crawl the final site under both `/` and `/zenfmt/` bases; reject broken fragments, case mismatches, absolute-root leaks, and external links not on the reviewed allowlist.
- Verify every book/ZDS HTML page links the correct PDF/source and every help mapping reaches a real heading.
- Verify every download card resolves to the exact versioned manifest asset, and that displayed target, archive type, byte size, SHA-256, requirement, and provenance agree with release data.
- At the side-by-side boundary, assert approximately 40/60 pane allocation and an output content box of at least 80 `ch`; at narrower widths, zoom, and larger font settings, assert that panes stack instead of compressing text.
- Confirm JavaScript-disabled access to homepage content, Book, ZDS, benchmark method, downloads, and security guidance.
- Verify no network request occurs during conversion after static assets are loaded.

156.4. Performance budgets

Budgets are measured in a pinned cold browser profile and recorded by release:

- non-WASM homepage shell is interactive without waiting for the engine;
- critical CSS plus initial JavaScript is at most 150 KiB compressed, with bundled fonts budgeted separately as stated earlier;
- each documentation route is at most 200 KiB uncompressed and 60 KiB compressed, excluding referenced images, which is what makes the figure extraction above a requirement rather than an optimisation;

- the complete `zenfmt.wasm` is at most 2 MiB raw and 512 KiB compressed. The first `ReleaseSafe` build of the full nineteen-reader bundle measured 1,728,110 bytes raw, 561,321 gzipped, and 388,066 Brotli-compressed, so the budget is that measurement plus about fifteen percent rather than the 25 MiB ceiling this record originally carried. The original figure was a guess made before the module existed and was fifteen times the real size; a regression that doubled the module would have passed it silently. Any exception requires an amendment with format-level size attribution;
- desktop reference engine instantiate median is at most 1 second and the mobile-emulation p75 is at most 2.5 seconds on the designated profiles;
- no long task over 50 ms is caused by conversion on the main thread;
- homepage layout shift is at most 0.1 and converter controls do not move when WASM becomes ready;
- documentation pages do not fetch the WASM module until the visitor activates a converter entry point;
- asset-size, parse/startup, and representative conversion regressions over 10% from the accepted 0.2.0 baseline fail the comparison gate unless the change records the reason.

These are delivery budgets, not hand-entered dashboard claims.

156.5. Release acceptance matrix

Gate	Pass condition
compile	Zig 0.16.0 builds the `wasm32-freestanding` artifacts from a clean checkout, and every engine, support, and format library compiles and passes its parity fixtures with a 32-bit `usize`.
authority	Import audit proves the capability-minimal module — an empty import set, the exact export set, no debug or name sections, a bounded memory maximum, and a stack below the data segment; CSP/network inventory is clean.
parity	All representative formats agree with native artifact/ resource/ report/manifest expectations.
interaction	All first-run, keyboard, worker lifecycle, and error recovery tasks pass in the gating browser, and the manual Safari passes are recorded.
documentation	Homepage, Book, ZDS, benchmark, download, and security routes exist with valid HTML, links, search, PDF, and source mapping.
benchmark	0.2.0 reference data is pinned, reproducible, correctness-gated, current, and clearly separates native/browser lenses.
accessibility	Automated and manual WCAG 2.2 AA review passes with no critical or serious issue.
performance	Shell/WASM/startup/main-thread/layout budgets pass.
release	WASM is present as a standalone target and complete bundle; every artifact version/digest agrees; attestations/

Gate	Pass condition
	checksums publish; Pages deploys the tag's commit; and public post-deploy conversion/download/help/PDF smoke tests pass.

157. GitHub Pages and Release 0.2.0

157.1. Pages workflow

Pull requests build and test the complete site, upload a review artifact, and never deploy. Pushes to `main` may deploy documentation only after site gates pass. The 0.2.0 tag workflow builds immutable release artifacts first; Pages for the release must consume the tag commit and released WASM digest rather than rebuilding an untracked variant. Checked-in benchmark data names the implementation revision it measured, and the version gate permits only benchmark results and ZDS lifecycle metadata between that revision and the tag.

Deployment uses GitHub's supported custom Pages workflow with configure, upload-artifact, and deploy actions; `pages: write` and `id-token: write` exist only on the deploy job. The `github-pages` environment protects production. Concurrency cancels superseded branch builds but never interrupts an active tag release after publication begins.

Post-deploy smoke tests use the public Pages URL and verify:

- root, book, ZDS, benchmark, download, security, PDF, and WASM responses;
- `application/wasm` or the adapter's documented safe fallback behavior;
- a real example conversion and native-known artifact digest;
- no outbound conversion request;
- theme persistence and the System default when no theme is stored;
- help links from a successful conversion and a forced error;
- direct downloads for WASM, every CLI target, Python, and source;
- standalone/archive/deployed WASM digest identity;
- release/version/commit/digest agreement.

157.2. Unified release artifacts

Release 0.2.0 advances the monorepo version once. The existing supported CLI archives, seven Python wheels, Python source distribution, and source archive are rebuilt at 0.2.0 alongside:

- `zenfmt-0.2.0-wasm32-freestanding.tar.gz`, the complete browser bundle;
- `zenfmt-0.2.0-wasm32-freestanding.wasm`, byte-for-byte identical to the module inside that bundle and available as its own release target asset;
- `zenfmt-book-0.2.0.pdf` plus stable Pages alias;
- `SHA256SUMS`, provenance attestations, SBOM/artifact manifest, and release notes that state browser limits and support.

Each ZDS PDF remains available from its stable GitHub Pages address. Individual ZDS PDFs are not included in the GitHub Release asset set.

The release workflow verifies the WASM archive in a clean temporary static server and real browsers, verifies the PyPI package after publication, verifies CLI archives, publishes and verifies `@insanai/zenfmt`, then creates the GitHub release. The npm package version must equal the tag and the native package version. Pages deployment follows a successful release and references

its URL. Partial publication stops further steps and is repaired by completing the same version; immutable PyPI or GitHub assets are never silently overwritten.

No release secret is available to the Pages runtime or artifact. PyPI uses Trusted Publishing where configured; no token appears in documentation, repository files, logs, or site output.

158. Operational Considerations

The site is static and has no application server to operate. Operational work is asset/version consistency, browser compatibility, dependency/toolchain review, benchmark freshness, link health, and rollback.

- Supported browsers are the current and previous stable Chromium, Firefox, and Safari/WebKit releases that implement WebAssembly, modules, workers, file APIs, and transferable buffers. Capability detection produces a useful fallback rather than browser-name sniffing.
- A Pages rollback redeploys a previously successful artifact and does not mutate the 0.2.0 release. Stable docs URLs continue to work.
- A severe WASM issue can disable the converter with a checked-in site flag while leaving Book, ZDS, downloads, and security guidance available. The flag must explain the reason and point to a safe released CLI/Python version.
- Dependency update automation may propose Zig, Typst, uv/Python packages, and Actions changes, but semantic HTML snapshots, WASM parity, and release reproducibility decide acceptance.
- Benchmark competitors remain pinned. Updating a competitor is an explicit benchmark change and regenerates all affected comparisons.
- The site contains a version selector only when multiple maintained documentation versions exist. Until then it shows the current version and a release-history link, avoiding a nonfunctional control.

159. Delivery Plan

159.1. Phase 0: toolchain determinism

- Vendor the Typst text fonts and `@preview` packages, pin the exact Typst patch version, and pass explicit creation timestamp, font path, and package path on every invocation.
- Prove a documentation build is byte-identical across two runs and across the supported development platforms, before any HTML work depends on it.
- Add the build revision option and the corpus digest manifest.

Exit: `zig build docs` is reproducible, and every later gate can be believed.

159.2. Phase 1: pure WASM boundary

- Refactor byte/memory conversion away from host path/thread construction, with the host arms eliminated at compile time rather than skipped at run time, so the absence of filesystem authority is a property the compiler enforces and the import audit can prove.
- Fix 32-bit size arithmetic and keep it fixed with a compile gate.
- Add the WASM ABI, allocator/result ownership, capability data, and import audit.
- Build the shipping and diagnostic artifacts, and establish native parity fixtures covering every reader in the default bundle.
- Record actual size/startup/memory baseline and address unsupported standard library calls without changing document semantics.

Exit: every default format compiles and representative conversions match native output under browser-profile limits.

159.3. Phase 2: browser API and playground

- Add the ES and worker adapters, declarations, Elm-style host diagnostics, cancellation, timeout, recycle, and browser tests.
- Build the converter workspace, read-only output, details, copy/download, responsive states, theme, privacy, and help ladder.
- Complete security review, CSP/network inventory, keyboard, screen-reader, and performance passes.

Exit: a clean static server can complete and recover from conversions in all supported browsers without document network traffic.

159.4. Phase 3: project site and documentation

- Build the shared site shell and route map.
- Export the book per chapter and ZDS per record to semantic HTML and PDF.
- Add navigation, local TOC, search, anchors, help map, source/PDF links, stable aliases, and responsive documentation layouts.
- Add site generation, link/semantic/accessibility tests, and local preview.

Exit: every published help path is reachable from the homepage and every generated route passes under both local and GitHub Pages base paths.

159.5. Phase 4: benchmark and release

- Pin competitor versions/digests and implement correctness-first browser benchmarking.
- Generate current native/browser/quality data, full dashboard, book chapter, and homepage summary from one schema.
- Integrate protected Pages workflow and unified release artifacts.
- Rehearse 0.2.0 from a clean tag candidate, publish, deploy, and run public smoke tests.

Exit: all acceptance gates pass and ZDS 0015 can move to `committed`.

160. Alternatives Considered

160.1. WASI in the browser

Rejected for 0.2.0. A WASI shim supplies filesystem-shaped capabilities the browser API neither needs nor wants. `wasm32-freestanding` is directly supported by Zig and makes the import audit small.

160.2. Emscripten or another compiler toolchain

Rejected. Zig already generates the required target out of the box. Another compiler/runtime would enlarge the toolchain, generated glue, and supply chain without providing conversion semantics.

160.3. Main-thread conversion

Rejected for the site. Even fast medians cannot guarantee a hostile or large document will not block input, rendering, and accessibility. The public module can be instantiated directly by expert callers, but the first-party site uses a worker.

160.4. WebAssembly threads

Rejected for 0.2.0. They require cross-origin isolation and increase browser, allocator, determinism, and deployment complexity. One conversion worker provides responsiveness; parallel parsing must be benchmark-justified later.

160.5. A JavaScript framework and npm build

Rejected. The site is a small static information architecture plus one worker application. Standards-based ES modules and CSS are sufficient, while Zig and uv-managed Python remain the build systems. Runtime JavaScript is the necessary browser adapter, not a second project-management ecosystem.

160.6. A single-page application

Rejected. Book/ZDS pages need stable URLs, semantic HTML, direct linking, search-engine discoverability, printable content, and useful no-JavaScript behavior. Static multi-page output matches GitHub Pages and documentation use.

160.7. Reauthoring the book in Sphinx or Markdown

Rejected. Typst already owns the book's diagrams, data-driven benchmark chapter, and PDF typography, and Typst 0.15 can emit semantic HTML/bundles. The site adopts Sphinx's navigation ergonomics without duplicating sources or adding a new authoring language.

160.8. Rendering converted Markdown as HTML

Rejected for security and clarity. The requested result is a read-only Markdown code window. Rendering would create a second sanitizer/security surface and blur the difference between source output and interpretation.

160.9. Opening output in a browser popup

Rejected. Popup blockers, focus changes, mobile behavior, and lost context violate the ergonomic goal. "Another window" is implemented as a clearly separate adjacent output panel in the page, stacking on small screens.

160.10. Running competitor benchmarks in every visitor's browser

Rejected. It would download large third-party artifacts, consume resources, produce incomparable devices, create license/cache concerns, and surprise visitors. The public dashboard uses pinned reference runs and offers explicit reproduction instructions.

160.11. One blended native/WASM headline

Rejected. Native process startup, warm library calls, WASM download/compile, and warm WASM execution answer different questions. The dashboard keeps native and browser lenses separate and labels cold/warm boundaries.

160.12. Third-party analytics and hosted fonts

Rejected. They weaken the promise that browser conversion is local, add network and privacy ambiguity, and are unnecessary for product correctness.

160.13. Publishing only the WASM file

Rejected. A raw module without a version-checked adapter, declarations, ownership contract, examples, checksum, and clean-browser tests is not a complete developer release.

161. Known Gaps Outside This Release

Implementing this record does not close every gap it touches, and listing the ones it leaves open is part of not overclaiming.

- Six readers — HTML, Markdown, AsciiDoc, reStructuredText, CSV, and plain text — have no format record of their own and are covered only by a section of ZDS
  2. The same six emit no facets, and most declare no preservation data

  version, which ZDS 0013′s delivery plan requires of every reader. Closing that needs its own records, one per format.
- The HTML reader silently discards script, style, embedded-graphic, form, and frame content. Release 0.2.0 gives it a diagnostic, because the browser playground makes web input the most likely first document a visitor tries; the reader's wider omissions still belong in a format record.
- Markdown remains the only writer, so the lowering and capability machinery is still validated against one implementation, and the binary-emission path declared by the plugin contract remains unreachable.

None of these blocks the browser target or the site. They are recorded here so this record's `committed` state is not mistaken for their completion.

162. Open Questions

There are no implementation-blocking open questions. Exact microcopy and final illustration details may be refined during visual review if they preserve the semantic tokens, original identity, accessibility, performance budgets, wireframe hierarchy, and interaction contracts in this record. A change to target, browser authority, routes, output safety, benchmark method, theme behavior, artifact set, or acceptance gates requires an amendment.

163. Acknowledgements

Authored by Vikrant Rathore with assistance from Ronak Rathore. The direct browser-conversion interaction takes inspiration from Firecrawl's AnyDoc demo; the documentation ergonomics take inspiration from Sphinx-style manuals; and the bold editorial hierarchy takes inspiration from Notion. The download information hierarchy takes inspiration from Zed's version-first, platform-oriented download page. zenfmt's design, copy, components, visual tokens, wireframes, and implementation remain its own.

164. References

- ZDS 0001, **The Zen Discussion Process**.
- ZDS 0002, **zenfmt: Architecture and Implementation**.
- ZDS 0013, **Layered Document IR and Writer Lowering**.
- ZDS 0014, **The zenfmt Python Library: API and Implementation**.
- Zig 0.16.0 language reference, WebAssembly: ziglang.org/documentation/0.16.0/#WebAssembly.
- Zig platform support, including `wasm32-freestanding`: ziglang.org/learn/platform-support.
- Typst bundle documentation: typst.app/docs/reference/bundle.
- Typst 0.15 release notes and HTML/bundle changes: typst.app/docs/changelog/0.15.0.
- GitHub Pages custom workflows: docs.github.com Pages custom workflows.
- MDN, `WebAssembly.instantiateStreaming`: developer.mozilla.org WebAssembly API.
- W3C, Web Content Accessibility Guidelines 2.2: w3.org/TR/WCAG22.
- Firecrawl AnyDoc browser/WebAssembly reference: github.com/firecrawl/anydoc.
- Pandoc browser WebAssembly application: pandoc.org/app.

- Zed download page: zed.dev/download.
- Steve Krug, **Don’t Make Me Think, Revisited**.
- Daniel Kahneman, **Thinking, Fast and Slow**.