Zen Working Group
Request for Discussion and Implementation Record

ZDS 0013
Architectural Specification

Layered Document IR and Writer Lowering

STATE
committed

CREATED
2026-08-07

LAST UPDATED
2026-08-07

INTENDED STATUS
Committed

AUTHORS
Zen Contributors <team@insan.ai>

DISCUSSION

A layered semantic IR with sparse facets and provably deterministic writer lowering, superseding the AST and writer sections of ZDS 0002

LABELS
architecture, ast, ir, writers, formats

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

104. Abstract

zenfmt reads nineteen formats and writes one. The next writers, HTML, DOCX, Typst, and the presentation and spreadsheet families, need information that Markdown discards: named styles, page geometry, sheet coordinates, tracked revisions, provenance. This record specifies IR version 2, a layered design that carries that information without inflating the flow representation that every conversion touches.

The design has four parts. First, a **semantic kernel**: the current flat preorder struct-of-arrays tree, retained because its central property, subtree contiguity, is proved here as Lemma 1, with its coordinated tag switches replaced by one genuine comptime schema table. Second, **sparse facets**: typed stand-off annotations (provenance, style, layout, grid, revision) bound to nodes through lazy entity identities, costing nothing when absent. Third, **extension nodes**: namespaced constructs with a mandatory source-neutral fallback subtree, so a writer that does not understand an extension still has something honest to emit. Fourth, **writer lowering**: each writer declares its capabilities at compile time, and a bottom-up dynamic program selects among exact emission, declared degradations, and refusal, under a lexicographic loss cost that Theorems 4 and 5 prove yields a unique, deterministic optimum.

The record was prompted by an external proposal (`zenfmt_advanced`) to rebuild the IR as a whole-document e-graph with equality saturation. The proposal is evaluated here and adopted only where it improves zenfmt. Its storage instincts already match the shipped design; its central idea is rejected on mathematical grounds: Theorem 3 shows that lossy writer alternatives are strict degradations, and a strict degradation cannot be an element of any equivalence relation, so e-graph union over them is unsound by construction. E-graphs are retained in one restricted role, as an optional optimizer over semantics-preserving rules, gated on a measured 15% end-to-end win.

This record supersedes the AST, builder, transform, and writer sections of ZDS 0002 and preserves everything else in it. zenfmt is pre-release with no external consumers, so IR v2 replaces IR v1 directly: the kernel is rewritten in place and every reader and writer is rewritten against it, with no dual-track staging and no compatibility layer. The rewrite has landed and every acceptance gate in Operational Considerations holds; this record is committed and descriptive of the current system.

105. Introduction

A document converter is a compiler. ZDS 0002 built the front half of one: a shared tree, nineteen front ends, one back end. The tree was designed for the back end it had, GitHub-Flavored Markdown, and it is honest about that. A DOCX paragraph style survives only as a preservation blob in the manifest. A PDF page position survives only long enough to decide paragraph boundaries, then is discarded. An XLSX formula survives as its cached value. For a Markdown target, these are the right losses. For a DOCX, HTML, or Typst writer, they are self-inflicted wounds: the reader had the information and threw it away before any writer could ask for it.

The multi-level IR literature reached the same conclusion for ordinary compilers: lowering too early destroys information that later stages need, so a compiler should preserve high-level semantics first and lower toward target-specific form as late as possible (Lattner et al., CGO 2021). This record applies that principle to documents, with one important difference. Compiler lowering is semantics-preserving; document lowering usually is not. Writing underlined text to Markdown loses the underline no matter how the writer spells it. The interesting structure is therefore not an equivalence, it is an ordering, and the machinery in this record is built on that ordering.

105.1. Relationship to other records

This record supersedes four parts of ZDS 0002: the sections *The Document AST*, *Traversal and Rewriting Algorithms* (the builder and the rebuild transform), and *The Markdown Writer*'s implicit lowering decisions. It preserves, and where noted repairs, everything else ZDS 0002 specifies:

bundle isolation, the `convert` API surface, format-library boundaries, the diagnostics philosophy, the adjacent manifest requirement, the coding standard, and the security limits.

The format records ZDS 0003 through 0012 remain the per-plugin ledgers. As each reader is rewritten, its record gains facet columns in its mapping table: which constructs populate which facets, in addition to which kernel nodes they produce. ZDS 0002 is amended alongside this record to reference it and to state precisely what its "one comptime schema" is today (coordinated exhaustive switches whose coverage the compiler enforces but whose agreement is by hand).

105.2. The triggering proposal

`zenfmt_advanced` is a captured search-result printout proposing a "pointerless, flat-array E-graph converter": every document node hash-consed into a `std.MultiArrayList`, equivalences tracked by union-find, and format conversion performed by rewrite rules that union a matched node with its translated form. Its storage instincts are sound and already shipped in zenfmt: struct-of-arrays layout, 32-bit typed indices, arena lifetime, linear scans over tag columns. Its central mechanism is not sound for documents; the Problem Statement and Alternatives Considered say precisely why. What the proposal gets right stays, because it already shipped in IR v1; its cost-based extraction survives, transformed, as the lowering planner; its equality machinery is excluded by Theorem 3.

106. Terminology and Scope

- **kernel**: the semantic document layer, the ordered forest of blocks and inlines with payloads, attributes, and metadata.
- **facet**: a typed, optional annotation record bound to a node through an entity identity, stored outside the kernel node arrays.
- **entity**: a stable logical identity (`EntityId`) shared by a kernel node and its facets, assigned lazily on first facet attachment.
- **resource**: bytes or an external reference carried alongside the document: images, embedded media, fonts. Generalizes the v1 media store.
- **extension node**: a namespaced kernel node owned by a plugin, carrying a mandatory fallback subtree of ordinary kernel nodes.
- **lowering**: a writer's translation of one kernel construct into target operations: exact, degraded, or refused.
- **loss set**: the structured record of what a chosen degradation discards.
- **choice DAG**: the bounded, acyclic set of lowering alternatives the planner considers for one node.
- **EMU**: English Metric Unit, 1/914400 inch, the OOXML drawing unit and the canonical coordinate unit of `LayoutFacet`.
- **e-graph**: a data structure representing a congruence relation over terms; **equality saturation** grows one to a fixed point under rewrite rules and extracts one term by cost (Tate et al., POPL 2009; Willsey et al., POPL 2021).

In scope: the IR v2 data model, its invariants and complexity bounds, the lowering planner, the formal loss taxonomy, the clean-break implementation plan, and the core contract repairs that the rewrite must carry. Out of scope: per-format mapping details (those stay in ZDS 0003 through 0012), the design of any individual new writer, and editing-grade source round trips, which remain a non-goal of the project.

107. A Formal Model of Lossy Conversion

The definitions here are small, but they carry the whole record. Every design decision that follows is an appeal to one of them.

107.1. Documents and observations

Definition 1 (Document). A document D is a finite ordered forest of nodes. Each node n carries a tag $\tau(n)$ from the kernel schema, an optional typed payload, optional attributes, and an ordered (possibly empty) sequence of children. $\mathbb{D}$ denotes the set of documents valid under the kernel schema and its invariants.

Definition 2 (Observation set). $\mathrm{obs}(D)$ is the set of semantic observations of D: every judgment a reader of the rendered document could make that the kernel schema is able to express. Text content and order, structural containment, emphasis and other span semantics, list kinds and numbering, table geometry, link targets, metadata values, and, when facets are present, the facet judgments. Two documents mean the same thing exactly when their observation sets are equal.

obs is deliberately abstract. The record never needs to enumerate it; it needs only that each kernel construct and facet contributes identifiable elements to it, so that set inclusion between observation sets is well-defined. This is the same move operational semantics makes with observational equivalence: define equality by what can be distinguished, not by syntax.

Definition 3 (Fidelity preorder). For $D, D' \in \mathbb{D}$, write $D' \preceq D$ ("D' is at most as faithful as D") if and only if $\mathrm{obs}(D') \subseteq \mathrm{obs}(D)$. Write $D' \prec D$ for the strict form, $\mathrm{obs}(D') \subsetneq \mathrm{obs}(D)$.

The relation $\preceq$ is reflexive and transitive because set inclusion is, so it is a preorder on $\mathbb{D}$. It is not antisymmetric on syntax, since two spellings can have equal observation sets; that quotient is exactly what the next definition names.

107.2. The five outcomes of conversion

Every act a converter performs on a construct is one of five things. The taxonomy is exhaustive by construction: either meaning is preserved (equality, normalization), reduced (degradation, and its special case omission), or nothing is emitted (refusal).

Definition 4 (Loss taxonomy). Let a conversion step take D to an output whose kernel reading is D'. The step is:

- **equality** if $\mathrm{obs}(D') = \mathrm{obs}(D)$;
- **normalization** if $\mathrm{obs}(D') = \mathrm{obs}(D)$ and D' is the canonical form of D under a convergent rewrite system (Definition 5);
- **degradation** if $D' \prec D$; its **loss set** is $\mathrm{loss}(D, D') = \mathrm{obs}(D) \setminus \mathrm{obs}(D')$, which is nonempty by definition;
- **omission** if the step is a degradation that removes a subtree entirely, so $\mathrm{loss}(D, D')$ contains every observation the subtree contributed;
- **refusal** if no output is produced and a diagnostic is produced instead.

Definition 5 (Convergent normalization). A rewrite system R over $\mathbb{D}$ is **semantics-preserving** if $D \underset{R}{\to} D'$ implies $\mathrm{obs}(D') = \mathrm{obs}(D)$; it is **convergent** if it is terminating and confluent, in which case every document has a unique normal form $\mathrm{nf}_R(D)$.

zenfmt already performs normalization without naming it: the builder coalesces adjacent text nodes and splits whitespace into `space` nodes; the Markdown writer emits one canonical spelling per construct. Definition 5 makes the obligation explicit: a normalization rule must be proved semantics-preserving, and the rule set must be convergent so the canonical form is unique. Termination plus local confluence suffices:

Theorem 1 (Newman). A terminating, locally confluent rewrite system is confluent, and therefore every element has a unique normal form.

Newman's classical result (Annals of Mathematics, 1942) is cited, not reproved. Its practical force here: zenfmt's normalization rules each strictly decrease the measure $\mu(D) = (\text{node count}, \text{nesting depth})$ in lexicographic order, so the system terminates, and local confluence is checked case by case over the finite rule set. A convergent system computes in bounded linear passes what equality saturation would compute by growing an e-graph to a fixed point, which is the first half of the argument against saturation. The second half is stronger.

107.3. Why lossy alternatives cannot be equalities

Theorem 2 (Degradation is asymmetric). If $D' \prec D$ then not $D \preceq D'$. In particular $\prec$ is irreflexive and asymmetric.

Proof. $D' \prec D$ means $\mathrm{obs}(D') \subsetneq \mathrm{obs}(D)$: inclusion holds and some observation $\omega \in \mathrm{obs}(D)$ has $\omega \notin \mathrm{obs}(D')$. If also $D \preceq D'$, then $\mathrm{obs}(D) \subseteq \mathrm{obs}(D')$, so $\omega \in \mathrm{obs}(D')$, a contradiction. Irreflexivity is the case $D' = D$. □

Theorem 3 (No equivalence relation contains a strict degradation). Let $\sim$ be any equivalence relation on $\mathbb{D}$ that respects observations, meaning $D \sim D'$ implies $\mathrm{obs}(D) = \mathrm{obs}(D')$. Then no pair with $D' \prec D$ satisfies $D \sim D'$. Consequently, placing a degraded alternative in the same e-class as its source either violates the observation discipline or silently redefines obs to forget the lost observations.

Proof. Suppose $D' \prec D$ and $D \sim D'$. By the observation discipline, $\mathrm{obs}(D) = \mathrm{obs}(D')$. By Definition 3, $\mathrm{obs}(D') \subsetneq \mathrm{obs}(D)$, so $\mathrm{obs}(D') \neq \mathrm{obs}(D)$. Contradiction. The only escape is to weaken obs until the lost observations are no longer observations, which is precisely the loss-hiding zenfmt exists to prevent. □

The damage does not stay local. An e-graph maintains a **congruence**: if children are equal, parents built from them are equal. One unsound union therefore propagates upward through every containing context. Merging `underline` with `emphasis` in one span makes every document containing that span "equal" to a version with the underline erased, and the extraction step then chooses among them by a scalar cost with no record that a choice discarded meaning. Equality saturation is the right tool when the rule set really is semantics-preserving and non-confluent, so that rule order matters and the e-graph explores all orders at once (Tate et al., POPL 2009). Writer degradation satisfies neither hypothesis: the rules are not semantics-preserving (Theorem 3), and the deterministic policy below makes order irrelevant by construction.

What lossy lowering actually needs is machinery for **choosing along a preorder while recording the descent**. That is an optimization problem over $\preceq$ with a cost that prices each loss set, and it has a classical deterministic solution: minimum-cost tree covering by dynamic

programming (Aho and Johnson, JACM 1976; Fraser, Hanson, and Proebsting, 1992). The Writer Lowering section builds exactly that and proves it deterministic.

108. Problem Statement

Three gaps, in increasing order of depth.

The kernel cannot carry what future writers need. The v1 AST has no home for named styles, page or slide geometry, sheet coordinates, formulas, tracked revisions, comments, or per-node provenance. Readers that see this information today either discard it or serialize it into per-plugin preservation blobs that only the same plugin can reuse. A DOCX-to-DOCX conversion through zenfmt would lose the style catalog that both ends understand, because the middle has nowhere typed to put it.

Writer lowering is implicit. The Markdown writer's decisions, `underline` becomes emphasis with a `STYLE DROPPED` note, nested tables flatten, are correct but live as imperative code. There is no declared set of alternatives, no stated policy for choosing among them, no guarantee that a second writer will make its choices the same way, and `--strict` is a per-writer reimplementation rather than a property of a plan. The loss taxonomy of Definition 4 exists as prose in ZDS 0002 (the lossiness tiers) but not as a contract the compiler can see.

The proposed alternative would hide losses instead of choosing them. The `zenfmt_advanced` proposal reaches for an e-graph, which Theorem 3 rules out as a home for degradations. Beyond the mathematics, the sketch itself is not production-ready. The defects matter because each one is a lesson the IR v2 design must not repeat:

Defect in the sketch	Why it matters
`ENodeId` and `EClassId` used interchangeably	After the first union they are different things: a node is a member, a class is a set. The sketch's `find` walks `parents` indexed by node id, so merged classes alias arbitrary members.
`unionClasses` never restores congruence	Real e-graphs must re-canonicalize parents and rebuild the memo after unions (the `rebuild` of Willsey et al.); without it the memo table returns stale classes and "instant deduplication" is silently wrong.
No e-class member lists or analyses	Extraction, the step that produces output, is impossible: there is no way to enumerate a class's members, and no cost data to choose among them.
No scheduler, budget, or termination argument	Equality saturation does not terminate in general; practical engines run under node, iteration, and time limits. The sketch has none, which for a converter of

Defect in the sketch	Why it matters
	hostile input is a denial-of-service invitation.
`child_count: u2` over `children: [2]EClassId`	The count type admits 3; the array holds 2. Documents need arbitrary fan-out; a paragraph with fifty inline runs does not fit a binary tree without an encoding the sketch never defines.
Text offsets hashed into node identity	Two equal strings at different pool offsets hash differently, so the memo never deduplicates the one thing documents actually repeat: text.
Hash-consing every node	Adds a random-access hash probe to every node construction. Prose documents share little identical structure; the probe is pure overhead on the common path, against zenfmt's measured append-only builder.
Pre-0.16 collection idioms	`std.ArrayList.init(allocator)` and managed `std.HashMap` in the form used no longer exist; the code as written does not compile on the Zig version zenfmt targets.
Uncited provenance, truncated code, unmeasured claims	The document is a search-result printout. Its "24 bytes per node" and cache claims are hypotheses. Per this record's own rule, no performance claim enters a ZDS without a checked-in result file.

109. Goals and Non-Goals

109.1. Goals

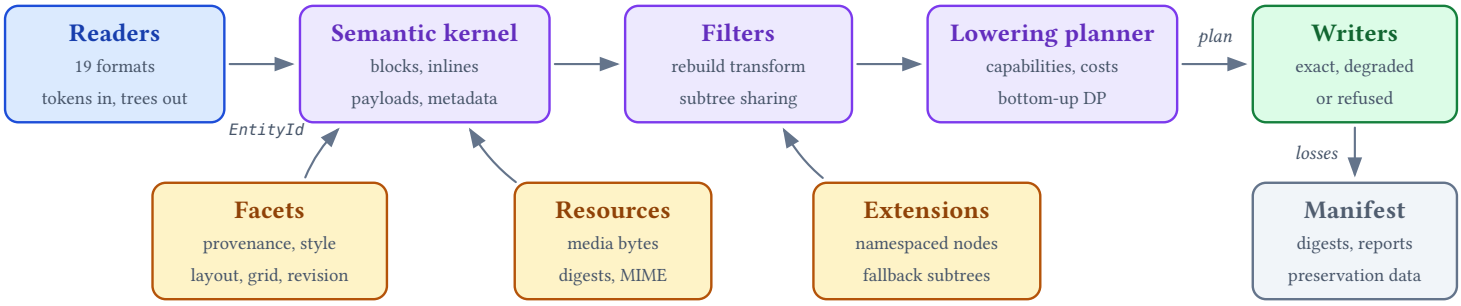
- Carry rich-document semantics (style, layout, grid, revision, provenance, resources) in typed, validated form, at zero cost to conversions that do not use them.
- Make writer lowering declarative: capabilities, alternatives, and refusals visible at compile time, selection deterministic and proved so.
- Report every chosen loss through the existing diagnostics system, from the selected plan only, with `--strict` a plan property rather than writer code.
- Keep the kernel free of format identifiers, exactly as ZDS 0002 requires of the engine.
- Preserve the measured strengths of IR v1: arena lifetime, subtree contiguity, bulk copying, bounded non-recursive traversal, append-only side tables.

- Adopt from `zenfmt_advanced` only the ideas that make zenfmt better, and record why the rest do not.
- State every new bound as a named limit and every complexity claim as a theorem or a benchmark obligation.

109.2. Non-Goals

- Editing-perfect source round trips. IR v2 improves fidelity; it does not promise byte-identical DOCX out for DOCX in.
- A general annotation graph. Facets bind to entities, entities to nodes; arbitrary node-to-node relations remain out of scope.
- Implementing the restricted `EquivalenceOptimizer`. This record defines its admission conditions; it ships nothing.
- Backward compatibility of any kind. zenfmt has no release and no external consumers; IR v1 is replaced in place, not staged beside, wrapped, or migrated from.

110. Design Overview



The pipeline shape of ZDS 0002 is unchanged: readers build one document, filters transform it, one writer emits it, the manifest records what happened. What changes is the payload each stage carries. Readers may now attach facets and register resources as they build. Filters see and preserve facets through the same rebuild transform. The writer no longer improvises its losses: a planner intersects the document against the writer's declared capabilities and hands the writer a selected plan whose every degradation is already priced, chosen, and reported.

The layering rule is strict and testable: **the kernel plus resources must render meaningfully with every facet table empty.** Facets refine; they never carry primary content. Axiom 1 below is the formal statement, and the validator enforces its checkable consequences.

111. The Semantic Kernel

111.1. What stays, and why it stays

IR v2 keeps the v1 storage design wholesale: two `std.MultiArrayList` node arrays in preorder, a UTF-8 text pool, a binary resource pool, append-only typed side tables, one arena per conversion, and `enum(u32)` indices for every edge. This is not inertia. The design's central property is exactly the one a document converter exercises most, and it deserves its proof.

Each stored node records `subtree_len`, the size of its subtree including itself. Children are found by hopping, and whole subtrees move as contiguous ranges.

Lemma 1 (Subtree contiguity). In a preorder layout of an ordered forest where each node n at index $i(n)$ stores $s(n) = |\text{subtree}(n)|$, the subtree of n occupies exactly the index interval $[i(n), i(n) + s(n))$. The children $c_1, \ldots, c_k$ of n begin at $i(c_1) = i(n) + 1$ and $i(c_{j+1}) = i(c_j) + s(c_j)$, and their intervals partition $(i(n), i(n) + s(n))$.

Proof. By structural induction on the subtree of n. If n is a leaf, $s(n) = 1$ and the interval $[i(n), i(n) + 1)$ contains exactly n. If n has children $c_1, \ldots, c_k$, preorder visits n first, then each child subtree in order. By the induction hypothesis, each c_j's subtree occupies $[i(c_j), i(c_j) + s(c_j))$ contiguously, so $i(c_{j+1}) = i(c_j) + s(c_j)$ and $i(c_1) = i(n) + 1$. The intervals are disjoint, consecutive, and their union with $\{i(n)\}$ is $\left[i(n), i(n) + 1 + \sum_j s(c_j)\right) = [i(n), i(n) + s(n))$. □

Three operational facts follow directly, and all three are load-bearing in the shipped code. Sibling iteration is the hop $i + s$, with no parent pointers. Skipping an unwanted subtree is one addition. Copying an unchanged subtree in the rebuild transform is one contiguous range copy per column, plus rebasing of cross-array indices, which is why an identity filter pass allocates nothing and a sparse edit costs time proportional to the changed spine rather than the document.

The node row layout is also retained: the block row's storage base remains 21 bytes (tag, payload index, attrs index, child range, `subtree_len`), an invariant the tests pin with `@sizeOf` assertions. The `zenfmt_advanced` sketch's fixed two-child rows are rejected here: Lemma 1 delivers arbitrary fan-out with contiguous ranges, which a binary encoding could only imitate by adding synthetic nodes.

111.2. What changes: one schema, actually one

ZDS 0002 describes "one comptime schema" that generates payload access, placement rules, validation, and dispatch. The shipped `core/src/payload.zig` is close but not that: it is a set of coordinated exhaustive `switch` functions (`blockContent`, `blockPlacementAllowed`, the per-tag validators) in one file. The compiler enforces **coverage**, since a new tag breaks every switch, but **agreement** between the functions is maintained by hand; nothing stops the content rule and the validator from asserting different child kinds for the same tag.

IR v2 replaces the coordinated switches with a single comptime table, one row per tag:

```
pub const BlockRow = struct {
    tag: BlockTag,
    payload: type,                 // void when the tag has no payload
    children: ChildKind,           // .none, .inlines, .blocks, .structural
    placement: Placement,          // where this tag may appear
    structural_parent: ?BlockTag,  // for private structure tags
};
pub const block_schema: []const BlockRow = &.{ ... };
```

Views, placement predicates, validator cases, visitor dispatch, debug names, and the writer capability machinery of Writer Lowering are all derived from this table by comptime iteration. A property that today is five agreeing switches becomes one row read five ways; drift is no longer expressible. The tag sets carry over from v1 with exactly one addition each: the 24 block tags and 20 inline tags keep their semantics, and a new `extension` tag joins each set for the namespaced constructs of Extension Nodes. The node set was designed for source-neutral document semantics and nothing in this record changes that judgment; what changes is how the rules about the tags are stated. The checked-in corpus goldens remain the behavioral contract for the unchanged tags.

111.3. Entities: identity only where identity is needed

Facets need a stable handle to the node they annotate, and that handle must survive the rebuild transform, which moves nodes to new indices. Stamping every node with an identity would tax the conversions that need none, and the acceptance gates forbid that. Identity is therefore lazy, and it lives entirely outside the node rows.

Definition 6 (Entity binding). `EntityId` is an `enum(u32)` assigned from a per-document counter. The binding is a partial injection from nodes to entities stored in a side table, the `EntityTable`: append-only rows of (node reference, `EntityId`) kept ordered by node index. Node rows carry no entity field; a node without facets appears in no table at all. A node receives an entity at most once, on first facet attachment; entities are never reused within a document.

The side table wins over a per-node column by arithmetic. A `u32` entity column costs 4 bytes per node on every conversion, which is 4 MiB of sentinels per million nodes on exactly the flow-only conversions that use none of it, and it would grow the block row past its 21-byte base. The `EntityTable` costs $12e$ bytes for e entities and zero when $e = 0$. Lookup from node to entity is binary search over the ordered rows, $O(\log e)$, or a linear merge when a pass walks nodes in order; lookup from entity to node indexes the same rows through a second sorted index.

Lemma 2 (Entity stability under rebuild). The rebuild transform emits its output as contiguous range copies over the intervals of Lemma 1, and for each copy the translation is one offset. Rebasing the `EntityTable` is a single ordered merge of its rows against those ranges, costing $O(e + r)$ for e entity rows and r copied ranges. Every surviving node keeps its entity and hence every facet bound to it; every dropped node's binding, and with it its facets, leaves the document; injectivity is preserved.

Proof. A copied range moves old indices $[a, a + l)$ to $[b, b + l)$ verbatim, so an old index x inside it maps to $b + (x - a)$, a bijection on the range. The ranges are disjoint sub-intervals of Lemma 1's partition and are emitted in ascending order, and the `EntityTable` is ordered by node index, so one forward merge visits each row and each range exactly once: $O(e + r)$. A surviving node lies in exactly one range, so its row is rewritten exactly once, and distinct nodes map to distinct new indices, preserving injectivity. A dropped node lies in no range, so the merge discards its row; facet rows keyed by that entity become unreachable and are excluded from emission. Nodes created by a filter have no rows and gain none. □

A conversion that attaches no facets therefore allocates no `EntityTable`, performs no entity lookups on any path, and keeps the 21-byte block row intact: the flow-only budget gate in Operational Considerations is met by construction, not by measurement after the fact.

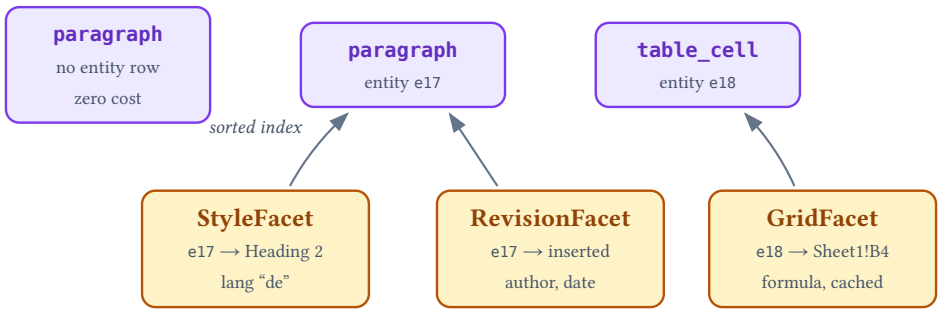
112. Sparse Facets

112.1. The catalog

Facets are stand-off annotation, the same separation the W3C Web Annotation model uses: the annotated thing and the annotation live apart, joined by identity, so the thing stays clean and the annotation stays optional. Each facet kind is a typed side table, append-only, with a sorted index from `EntityId` to row.

Facet	Carries	First producers
`ProvenanceFacet`	Source member and byte or logical range, producing plugin, confidence grade for heuristic structure (PDF paragraphs are guesses; DOCX paragraphs are facts).	all readers
`StyleFacet`	Named style, semantic role, language tag, direction, typed properties (the resolved ones, not raw format XML).	DOCX, ODT, HTML
`LayoutFacet`	Page, slide, or canvas identity; bounding box; transform; z-order; column membership; reading-order relation. Coordinates are normalized at read time to top-left-origin EMU (next subsection).	PDF, PPTX, ODP
`GridFacet`	Sheet and cell coordinates, value type, formula text, cached value, merge extents, row and column properties.	XLSX, XLS, XLSB, ODS
`RevisionFacet`	Insertions, deletions, comments, bookmarks, authorship, timestamps.	DOCX, ODT

The `ResourceStore` generalizes the v1 media pool: content bytes or an external reference, MIME type, BLAKE3 digest, pixel dimensions when known, and accessibility text. `image` payloads and future embedded objects hold a `ResourceId` instead of a private byte range, and the manifest's `media` array becomes the serialized view of this store.



112.2. One coordinate system

`LayoutFacet` coordinates are normalized at read time to a top-left origin and a single unit, the EMU, stored as `i32`. The alternative, tagging each facet with its source's unit, would force every paginated writer to carry an $M \times N$ conversion matrix and every mixed-source pipeline to reconcile units at emission time. Normalizing once, in the reader that already knows its source geometry, is strictly less code in the only place that has the context.

The unit is chosen by arithmetic, not preference. The candidate units either divide one another exactly or they do not:

$$1\ \text{pt} = 12700\ \text{EMU}, \quad 1\ \text{CSS px} = 9525\ \text{EMU}, \quad 1\ \text{mm} = 36000\ \text{EMU}$$

Every coordinate a DOCX drawing, PPTX slide, ODP frame, or CSS length names is therefore an exact integer in EMU. The reverse direction fails: a millipoint ($1/7200$ inch) is 127 EMU, so a millipoint normalization would divide every native EMU coordinate by 127 and round, corrupting exactly the sources the facet exists to serve. PDF user-space reals round to the nearest EMU, an

error of at most half an EMU, about 28 nanometers, far below rendering significance; a reader that needs the original reals records them in `ProvenanceFacet`. An `i32` EMU coordinate spans about ± 59 meters, comfortably beyond any page, slide, or canvas, and overflow at the boundary is a validation error, not wraparound. The top-left origin matches every supported source except PDF, whose bottom-up y axis the reader flips using the page height it already holds.

112.3. The facet axiom

> **Axiom 1 (Facet erasure).** For every document D with facet tables F, erasing all facets preserves kernel semantics: the kernel observations of (D, F) and $(D, \varnothing)$ are identical. Equivalently, $\mathrm{obs}(D, F) = \mathrm{obs}(D, \varnothing) \cup \mathrm{obs}_{\mathrm{f}}(F)$ where $\mathrm{obs}_{\mathrm{f}}(F)$ are the facet observations: facets only ever add observations, never alter or replace kernel ones.

This is an axiom in the design sense: it cannot be proved about arbitrary plugin behavior, so it is imposed on it, and its checkable consequences are enforced. The validator rejects a document whose rendering would depend on a facet: text content must live in the text pool, never in a facet; a facet may not be the only carrier of a link target or list structure; an entity must resolve to exactly one surviving node. A reader that wants a facet-like thing to affect flow must express it in the kernel, where every writer can see it.

The payoff is compositional: the Markdown writer's correctness is untouched by any facet any future reader invents, because by Axiom 1 the kernel it reads means the same thing with the facet tables ignored. Facet-aware writers opt in per kind through their capability declaration.

112.4. Cost model

Facet tables are append-only during reading and sorted by construction, because readers traverse sources in order; a final sort pass covers the exceptions, counted against the same bound. Lookup during writing is binary search, $O(\log m)$ for m facet rows, or a linear merge when the writer walks the document in order, $O(n + m)$ total. Every table is bounded by the new `max_facet_rows` limit, and a conversion with $m = 0$ performs no facet allocation at all. These are obligations, not hopes: the benchmark suite gains stage-separated measurements (Operational Considerations) and the acceptance gate pins the flow-only overhead.

113. Extension Nodes

Formats will always contain constructs the kernel does not model, and ZDS 0002's answer, per-plugin preservation blobs in the manifest, keeps them invisible to every other plugin. IR v2 adds a middle path that keeps the kernel closed while letting structure flow between consenting plugins.

An extension node is a kernel block or inline with three fields: an owner (reverse-DNS, the same namespace discipline as plugin ids), an extension name with a schema version, and a **mandatory** fallback subtree of ordinary kernel nodes. An optional reference ties it to plugin-owned preservation data in the manifest.

> **Axiom 2 (Fallback adequacy).** For every extension node x with fallback $f(x)$, the fallback is a degradation of the extension's meaning, never an unrelated rendering: $\mathrm{obs}(f(x)) \subseteq \mathrm{obs}(x)$, and the difference $\mathrm{obs}(x) \setminus \mathrm{obs}(f(x))$ is exactly what a non-understanding writer reports as loss.

The rules are mechanical. A writer that declares the extension namespace may consume the node and its preservation data. Every other writer lowers the fallback subtree as if the extension node

were not there, and the planner attaches the extension's loss set to that choice, so the report is automatic and uniform rather than re-invented per writer. The validator enforces the checkable part: the fallback is present, non-empty or explicitly declared empty with a reason, contains no extension nodes of the same owner (no fallback recursion past `max_depth`), and validates under the ordinary placement rules. Unknown payload bytes stay in the manifest; they never enter kernel storage as unvalidated format data.

114. Writer Lowering

114.1. Capabilities, declared at compile time

Each writer's descriptor grows a capability declaration, derived against the same schema table as everything else:

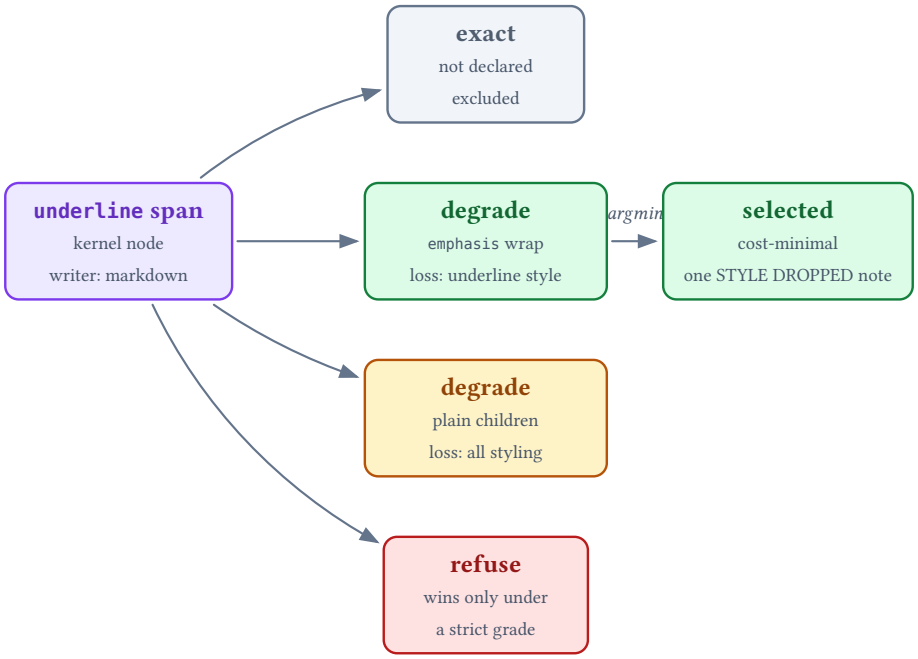
```
pub const capabilities = zenfmt.WriterCapabilities{
    .exact_blocks = &.{ .paragraph, .heading, .quote, .list, ... },
    .exact_inlines = &.{ .text, .emphasis, .strong, .code, ... },
    .facets = &.{},                      // markdown consumes flow only
    .extensions = &.{},
    .lowerings = &markdown_lowerings,    // declared degradation rules
    .refusals = &.{},                    // constructs it will not degrade
};
```

Comptime validation proves the declaration total: every kernel tag is exact, lowered, or refused, or the bundle fails to compile with a four-question error naming the uncovered tag. This is the writer-side mirror of the reader's mapping-table obligation from ZDS 0001, and it is what makes the loss taxonomy a contract instead of documentation.

114.2. The choice DAG

For each node the planner considers a bounded set of alternatives: the exact operation when declared, each applicable lowering rule's degraded subtree with its declared loss set, and refusal. Rules may produce kernel constructs that themselves need lowering, so alternatives chain; the chain is finite because each rule strictly descends the fidelity preorder (its loss set is nonempty by Definition 4) and `max_lowering_alternatives` bounds the width.

Alternatives are never materialized as trees. Each is a fixed-size `LoweringInstruction` descriptor of 16 bytes: an operation (emit exact, wrap in a tag, splice fallback, emit text), its target, a loss-set reference, and the rule identifier. The planner examines descriptors, not subtrees, so a rejected alternative costs its descriptor and nothing else, and no candidate ever allocates kernel nodes in the arena. Emission walks the selected plan and expands descriptors directly into the output stream. `max_lowering_work` therefore bounds descriptors examined, which is the true unit of planning work.



114.3. The cost order

Definition 7 (Loss cost). A lowering alternative's cost is the vector $c = (c_1, c_2, c_3, c_4, c_5, c_6) \in \mathbb{N}^6$ whose components count, in order: dropped semantic content; structural degradation; style, layout, revision, and metadata loss; emitted warnings; refusal pressure, nonzero only for refusal alternatives; and writer-local size or readability cost. Costs combine by componentwise addition $\oplus$ and compare by the lexicographic order $\prec_{\text{lex}}$; exact emission has cost $(0, \ldots, 0)$. Final ties break by the stable rule identifier, an `enum(u16)` fixed in the writer's source.

Lemma 3 (Total order). $\prec_{\text{lex}}$ on $\mathbb{N}^6$ extended by the rule-identifier tiebreak is a total order, and every finite nonempty set of alternatives has a unique minimum.

Proof. For vectors $a \neq b$ there is a first index k with $a_k \neq b_k$, and exactly one of $a_k < b_k$ and $b_k < a_k$ holds, so $\prec_{\text{lex}}$ totally orders distinct vectors. Distinct alternatives with equal vectors are separated by their rule identifiers, which comptime validation requires to be distinct. A total order on a finite nonempty set attains its minimum at exactly one element. □

Lemma 4 (Translation invariance). For all $a, b, c \in \mathbb{N}^6$: if $a \prec_{\text{lex}} b$ then $a \oplus c \prec_{\text{lex}} b \oplus c$.

Proof. Let k be the first index where $a_k \neq b_k$, with $a_k < b_k$. Adding c componentwise preserves equality at every index before k and preserves $a_k + c_k < b_k + c_k$ at k, since addition on $\mathbb{N}$ is strictly monotone. Hence the first difference of $a \oplus c$ and $b \oplus c$ is at index k and has the same sign. □

114.4. The planner and its guarantees

The planner runs bottom-up over the document in one reverse-preorder pass: by Lemma 1, a node's children are processed before the node, and each node's plan cost is

$$\mathrm{cost}(\mathrm{plan}(n)) = \min_{a \in \mathrm{alt}(n)} \left(\mathrm{cost}(a) \oplus \bigoplus_{m \in \mathrm{ch}(a)} \mathrm{cost}(\mathrm{plan}(m)) \right)$$

where $\mathrm{ch}(a)$ are the kernel nodes the alternative a emits or delegates. Exact-capability nodes short-circuit: when the tag is in the writer's exact set and no descendant requires lowering, the planner records the zero-cost plan without materializing alternatives, so a fully supported document, today's Markdown common case, pays one tag-set membership test per node.

Theorem 4 (Optimal substructure). With costs combined by $\oplus$ and compared by $\prec_{\text{lex}}$, the bottom-up recurrence computes, for every node, a plan of globally minimal cost for that node's subtree.

Proof. Structural induction on the subtree. For a leaf, the recurrence minimizes over $\mathrm{alt}(n)$ directly (Lemma 3). For an interior node, suppose some plan P for n has cost strictly below the recurrence's result. P chooses some alternative a and induces plans P_m for the children of a. By the induction hypothesis, $\mathrm{cost}(\mathrm{plan}(m))$ is at most $\mathrm{cost}(P_m)$ in $\prec_{\text{lex}}$ for each child m. Applying Lemma 4 once per child, with $\oplus$ commutative and associative on $\mathbb{N}^6$, substituting each P_m by $\mathrm{plan}(m)$ does not increase the total, so the recurrence's value at a is at most $\mathrm{cost}(P)$. The recurrence minimizes over all alternatives including a; contradiction. □

Theorem 5 (Determinism). For a fixed input document, writer, options, and limits, the selected plan is unique, and the emitted bytes and diagnostics are a pure function of those inputs.

Proof. At every node the candidate set is finite, bounded by `max_lowering_alternatives`, and nonempty, since refusal is always a candidate; by Lemma 3 its minimum is unique. By Theorem 4 the bottom-up pass visits nodes in a fixed order and depends only on already-fixed child plans. Emission walks the unique plan deterministically, and diagnostics are generated only from selected alternatives' loss sets, aggregated by the deterministic rules of ZDS 0002. No step consults time, randomness, or allocation addresses. □

Theorem 5 is the property the whole project leans on: it extends the byte-determinism promise of ZDS 0002, same input, same output, same manifest, across every future writer, including ones whose lowering is genuinely complicated. Planning work is bounded by $\sum_n |\mathrm{alt}(n)| \le n \cdot A_{\max}$ alternatives examined, each in constant time plus child-cost addition, so the planner is $O(n \cdot A_{\max})$ time and $O(n)$ space, with `max_lowering_work` as the hard runtime backstop against pathological rule interactions.

114.5. Strict mode and refusal

`--strict` becomes a plan predicate, not writer code, and the cost vector makes the predicate tunable. After selection and before any emission, the selected plan's aggregated cost c is tested against the requested grade:

Flag	Refuses when	Meaning
`--strict=content` (bare `--strict`)	$c_1 > 0$	Any dropped semantic content refuses; structural and stylistic degradation proceed, reported.

Flag	Refuses when	Meaning
`--strict=structure`	$c_1 + c_2 > 0$	Content loss or structural degradation refuses; style, layout, revision, and metadata losses proceed, reported.
`--strict=exact`	$c_1 + c_2 + c_3 > 0$	Any degradation at all refuses; the plan must be pure exact emission.

The predicate is well-defined because the plan's cost is the unique minimum of Theorem 5: the grade tests the best plan the writer has, so a refusal means no admissible plan exists under that grade, not that the planner chose badly. On refusal the writer never starts writing, so strict failures leave no partial artifact, preserving the transactional commit contract. Constructs in the writer's `refusals` set refuse in every mode and under every grade; this is for targets where degradation would be actively misleading, such as emitting a formula's cached value into a format whose consumers will recompute it.

115. Normalization, Not Saturation

Deterministic canonicalization already gives zenfmt unique normal forms wherever its rule set is convergent (Theorem 1): span flattening, adjacent text coalescing, empty-container dropping, attribute ordering. For a convergent system, a saturated e-graph followed by extraction computes the same normal form at strictly greater cost, so saturation can only earn its complexity where rules are genuinely non-confluent and semantics-preserving, the phase-ordering situation of Tate et al. No such rule set is known in zenfmt today.

A future `EquivalenceOptimizer` is therefore documented as optional, and admissible only under all of the following:

- It operates on one bounded subtree at a time, never the whole document.
- Every rule is semantics-preserving in the sense of Definition 5; rules expressing degradation are inadmissible by Theorem 3. In particular `strong` and `emphasis` interchange, heading-level changes, table flattening, and layout removal are invalid rules.
- It carries explicit node, iteration, work, and memory limits, and a correct rebuild step restoring congruence after unions, with the union-find bound of $O(\alpha(n))$ amortized per operation (Tarjan, JACM 1975).
- It ships behind differential tests: optimizer output must be observation-equal to the deterministic canonicalizer's output on the whole corpus.
- It ships only with a checked-in benchmark showing at least a 15% end-to-end improvement on a representative corpus. Absent that number, the deterministic canonicalizer stays.

116. Worked Mappings

One construct per family, traced end to end. Format records own the full tables; these rows exist to prove the layering exercises every facet kind.

Source	Kernel	Facets and resources
DOCX `w:ins` tracked insertion	The inserted runs as ordinary inlines; the paragraph reads as accepted text.	`RevisionFacet` on the run span's entity: kind insertion, author, timestamp. Markdown lowering drops it

Source	Kernel	Facets and resources
		with one aggregated note; a DOCX writer reconstructs `w:ins`.
DOCX named style "Heading 2"	`heading` level 2, resolved through the basedOn chain exactly as ZDS 0003 maps it.	`StyleFacet`: the style name, language, direction, so a style-aware writer can emit the catalog reference instead of a bare level.
PDF line placed at (72, 640) on page 3	`paragraph` content by the projection rules of ZDS 0011.	`ProvenanceFacet` with confidence "projected"; `LayoutFacet` with page 3 and the bounding box, so a paginated writer can restore placement and a flow writer can ignore it.
XLSX cell B4 with formula `=SUM(B1:B3)`	`table_cell` containing the cached value as text, exactly as today.	`GridFacet`: Sheet1!B4, numeric type, formula text, cached value. A spreadsheet writer re-emits the formula; Markdown keeps the cached value and reports the formula as carried but unused, not lost.
PPTX slide 4 title placeholder	`heading` from the title text, body placeholders as blocks.	`LayoutFacet`: slide 4 identity, placeholder geometry, z-order, so a presentation writer can rebuild the slide rather than a heading list.
HTML `<details>` disclosure widget	Extension node, owner `ai.insan.zenfmt.html`, fallback: `container` holding the summary as a paragraph then the content blocks.	No facet. A future HTML writer emits `<details>` from the extension; all other writers lower the fallback and report the interactivity loss per Axiom 2.
Markdown output (any input, flow only)	Consumes the kernel exactly as today; its capability set declares no facets.	By Axiom 1 the writer is provably unaffected by every row above; its checked-in corpus goldens are the rewrite's behavioral oracle.

The XLSX row shows a distinction the diagnostics gain from the taxonomy: information that is **carried but unused** by the selected writer, a formula riding in a facet Markdown ignores, is not a loss of the conversion, because it remains in the manifest-visible document; the note says so. Information the selected plan discards is a loss and is priced by Definition 7. Today both would be reported identically or not at all.

117. Core Contract Repairs

The rewrite touches every reader and the engine; the following repairs land with it as normative requirements, each anchored to a verified present defect. Hiding them inside implementation details would repeat the drift this record exists to stop.

1. **Manifest schema v2, typed preservation access.** Readers and writers of the same format family get typed access to exactly their preservation namespace, in both directions; today the codec carries data forward but no writer-side read path exists. Unknown namespaces

continue to round-trip canonically. Carried-but-unused facet tables serialize in two tiers: by default, one manifest entry per table holding its BLAKE3 digest, row count, and an `unused` marker, keeping the sidecar small; under `--preserve-facets`, the full facet rows serialize so a later same-family writer can recover them without the source document.

2. **Honest canonical JSON.** `core/src/json.zig` claims RFC 8785. It does not implement it: JCS orders keys by UTF-16 code units and serializes numbers as ECMAScript doubles, which cannot represent every 64-bit byte count the manifest may carry; zenfmt orders bytewise over UTF-8 and must keep exact integers. The claim is replaced by a self-contained "zenfmt canonical JSON profile v1": UTF-8, bytewise key order, no insignificant whitespace, exact integer spelling, shortest round-trip floats, a fixed escape set. The profile is specified, tested against goldens, and never called RFC 8785.
3. **Transactional path output.** The artifact, resources, and manifest stage to temporary names and the manifest publishes last, so an adjacent manifest never describes bytes that are not fully on disk. This formalizes the current commit order and extends it to the `ResourceStore`.
4. **Stream output completeness.** Direct stream output reports whether the stream received nothing, a partial prefix, or the complete artifact, so a failed streamed conversion is distinguishable from an empty document.
5. **One diagnostics data model.** Text and JSON renderers consume one report structure; aggregation keys include the complete identity (code, consequence, directions); contexts deduplicate; direction titles render in both forms.
6. **Detection compares evidence.** Extension routing and content sniffing both run; when they disagree, content evidence routes and a `core.extension-mismatch` note names both findings. Today the extension wins silently, so `report.docx` containing RTF parses as DOCX and fails confusingly.
7. **Validation effective in ReleaseFast.** Invariants that guard memory safety or output correctness are error returns in `validate`, never `std.debug.assert`, which ReleaseFast deletes. Asserts remain for programmer-error checks whose failure ReleaseSafe catches in tests.
8. **Operational `InputMode`.** A reader declaring `.seekable` receives a file-backed input rather than a slurped byte slice, and the engine's spill-to-temp path for stdin becomes part of the tested contract. The adopted scope: the ZIP layer, which backs ten of the nineteen formats, reads the central directory and each entry through bounded windowed reads, so an archive is never held in memory beside its expanded content; the CFB and PDF readers keep whole-input access through an explicit one-line shim, documented as such; and for file-backed inputs whose extension already routes, content sniffing is skipped, so the extension-mismatch note applies to byte inputs and extensionless files.
9. **Generated documentation.** The diagnostic catalog, capability matrices, limit tables, and format lists in the book and `--help` are generated from the code-owned registries: the schema table, the bundle, and `Limits`. The book's reference chapter cannot drift from the compiler's truth.
10. **Benchmarks by stage.** The harness separates process startup, parsing, IR construction, filtering, lowering, and rendering, so a regression names its stage. Restated as standing policy: no performance claim enters any ZDS without a checked-in result file.

118. Limits

New named limits, following the existing discipline that every limit has a name, a default, a CLI override, and a recorded reason. Defaults are starting points to be revisited against the corpus before the record leaves discussion.

Limit	Default	Bounds
`max_nodes`	16 Mi	Total kernel nodes, blocks plus inlines, per document; a hostile input cannot grow the arena without bound.
`max_facet_rows`	1 Mi	Total facet rows across all facet tables; a facet bomb, one paragraph with a million revisions, is a report, not an allocation storm.
`max_resources`	256	Resource entries per document; renames `max_media_files` as the store generalizes.
`max_resource_bytes`	128 MiB	Total resource bytes; renames `max_media_bytes`.
`max_lowering_alternatives`	8	Alternatives the planner materializes per node; the choice DAG stays a bush, not a thicket.
`max_lowering_work`	64 Mi	Total alternatives examined per document; the hard backstop behind the planner's $O(n \cdot A_{\max})$ bound.
`max_reports_total`	16 Ki	Aggregated report groups per conversion; beyond it, a final report counts the remainder.
`max_decoded_text_bytes`	256 MiB	Decoded text pool size, distinct from `max_input_bytes`, so a small compressed input cannot decode into an unbounded pool.

119. Security Considerations

The threat model is unchanged: every input is hostile. The new machinery adds three surfaces, each closed by construction plus a limit.

Facet bombs. Facets multiply per-node data. `max_facet_rows` bounds the total; the sorted-index discipline means lookup cost cannot be degraded by adversarial ordering; Axiom 1 means a discarded facet table can never change what renders, so refusing at the limit is safe as well as honest.

Lowering work bombs. A document engineered so every node needs maximal alternatives costs $n \cdot A_{\max}$ planning steps by design; `max_lowering_work` converts a rule-interaction surprise into a `limit`-class refusal rather than a hang. The planner allocates from the conversion arena, so its memory rides the existing input-proportional budget.

Extension fallback abuse. Fallback subtrees validate under the ordinary placement rules and `max_depth`; same-owner nesting is rejected, so a fallback cannot smuggle unbounded recursion or unvalidated bytes into the kernel. Unknown extension payloads stay in the manifest under `max_plugin_data_bytes`, exactly as v1 preservation data does.

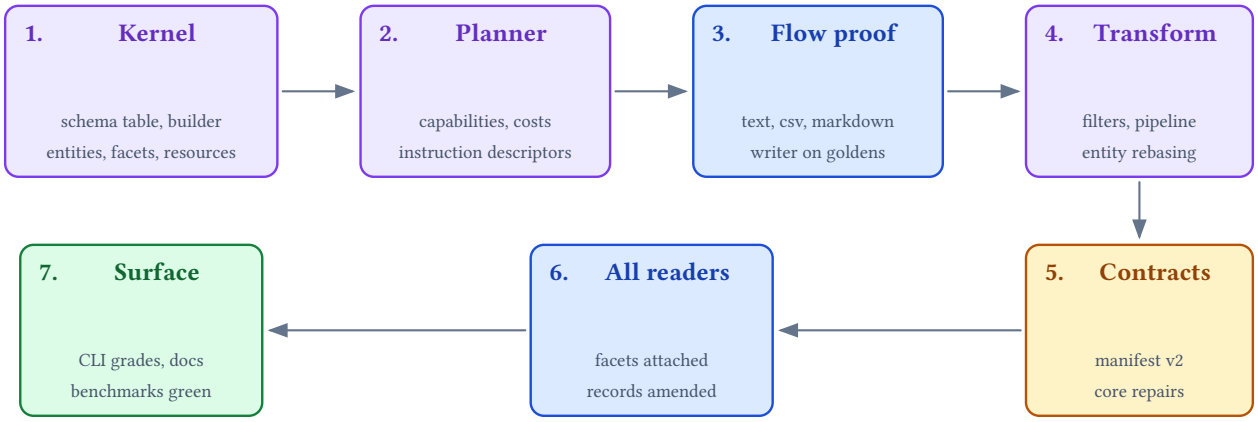
The ReleaseFast repair above is itself a security item: a validator whose checks vanish in the shipped optimization mode protects only the test suite.

120. Operational Considerations

120.1. Implementation plan: the clean break

zenfmt has no release and no external consumers, so nothing constrains the rewrite except its own gates. A dual-track staging (`core/src/ir2/` beside the current AST) was considered and rejected: it means two node schemas, two builders, two validators, and equivalence tests against a representation that would be deleted before the first release, all cost and no protection. IR v2 therefore replaces IR v1 in place. The kernel modules in `core/src/` are rewritten directly, and every reader and writer is rewritten against them in one sustained series of changes. The behavioral contract during the rewrite is not a parallel v1 build; it is the checked-in corpus goldens, the report snapshots, and the checked-in benchmark results, all of which predate the rewrite and none of which move.

The rules of the monorepo hold throughout: TigerBeetle coding standard (no recursion, every loop bounded, two or more asserts per non-trivial function, files under 1000 lines, functions under 70 lines), one arena per conversion, Elm-style diagnostics for every refusal, and `zig build fmt-check` plus `zig build test` green at every landing point.



1. Rewrite the kernel in place: `core/src/ast.zig` becomes the schema table and derived views; `builder.zig` gains the `EntityTable`; new `facets.zig` and `resources.zig` hold the five facet tables and the `ResourceStore`; the validator's invariants become error returns. Every file stays under the 1000-line rule; what does not fit splits by topic, as the format libraries already do.
2. Implement `core/src/lowering.zig`: capability declarations checked at comptime against the schema table, the choice DAG over `LoweringInstruction` descriptors, the bottom-up planner of Theorems 4 and 5, and the graded strict predicates.
3. Rewrite the flow path first and prove it: text, CSV, and Markdown readers plus the Markdown writer as capability declarations and lowering rules. Corpus goldens must remain byte-identical before anything else moves.
4. Rewrite the transform and pipeline on the new kernel: range-copy rebuild, entity rebasing per Lemma 2, zero-allocation identity passes.
5. Implement manifest schema v2 with the dual-tier facet serialization and writer-side preservation access, and land the core contract repairs.
6. Rewrite every remaining reader against the kernel, attaching its facet kinds: style and revision for DOCX and ODT; layout for PDF, PPTX, and ODP; grid for XLSX, XLS, XLSB, and ODS; extension nodes for HTML and EPUB; provenance everywhere; resources through the `ResourceStore`. Each format record's mapping table gains its facet columns in the same change.

7. Update the umbrella, default bundle, and CLI: `--strict={content,structure,exact}`, `--preserve-facets`, documentation regenerated from the code-owned registries, and the benchmark suite rerun with results checked in.

120.2. Acceptance gates

Each gate is a test or a checked-in measurement, not a judgment call. All of them hold in the implemented system; the suites and result files named below are the standing evidence.

Gate	Evidence
The schema table carries all 25 block and 21 inline tags, the v1 sets plus the two extension tags, and views, placement, validation, dispatch, and capability checks are all derived from it.	exhaustive schema test
Flow, layout, grid, revision, resource, and extension examples validate with no format identifier in any core field.	validator suite
Markdown output over the corpus is byte-identical to the checked-in goldens, and every selected loss is reported.	golden diff, report snapshots
A synthetic second writer exercises capabilities, alternative selection, graded strict refusal, and preservation recovery.	`tests/lowering.zig`
Identity filters allocate nothing; sparse edits stay proportional to the changed spine.	counting-allocator tests
Flow-only conversions stay within 15% of the checked-in benchmark results in peak memory and latency; absent facets allocate zero facet rows and no `EntityTable`.	stage-separated benchmark, checked in
Lowering work and alternatives stay bounded under hostile inputs.	adversarial corpus, limit refusal tests
All 19 readers have tree or facet goldens, malformed cases, allocation-failure tests, and fuzz targets on IR v2.	per-format suites

121. Alternatives Considered

Alternative	Why not
Keep the current IR unchanged	Every future writer beyond Markdown re-derives, per format pair, data the readers already had; preservation blobs make that possible only within one format family. The gap is structural, not incremental.

Alternative	Why not
Pointer-based tagged-union tree	Loses Lemma 1 and everything downstream of it: bulk copies, hop iteration, arena density. Rejected in ZDS 0002, and nothing has changed.
Whole-document e-graph (the proposal)	Unsound for the dominant use case by Theorem 3; costly for the common case, hash-consing overhead against near-zero sharing in prose; and the supplied sketch is additionally defective as tabulated in the Problem Statement.
Generic property graph	Maximally flexible and minimally checkable: placement rules, exhaustive dispatch, and the validator all dissolve into runtime schema checks. The compiler stops helping.
One IR per format, pairwise lowering	The pandoc argument in reverse: n formats need n^2 translations. The shared kernel is the whole point of the architecture.
Facets inside node payloads, fat nodes	Every flow conversion pays for every rich field; violates the flow-only budget gate immediately. Sparsity is the design.
Layered semantic IR with facets and lowering (this record)	Adopted: keeps the proved kernel, adds meaning where writers need it, prices every loss, and stays deterministic by Theorem 5.

122. Open Questions

None remain. The five questions raised in the first draft of this record were resolved by review, and each resolution is now normative in the body:

- Entity storage: the sparse `EntityTable` side map, no per-node column (Definition 6 and the arithmetic that follows it).
- Layout coordinates: normalized at read time to top-left-origin EMU as `i32`, with source reals preserved in `ProvenanceFacet` where a reader needs them (Sparse Facets, One coordinate system). The review proposed millipoints; the divisibility argument recorded there corrects that to EMU.
- Planner representation: `LoweringInstruction` descriptors, never materialized candidate subtrees (Writer Lowering, The choice DAG).

- Carried-but-unused facets: digest plus row count by default, full rows under `--preserve-facets` (Core Contract Repairs, item 1).
- Strict grading: `--strict={content,structure,exact}` as predicates over the aggregated cost vector (Writer Lowering, Strict mode and refusal).

123. References

- ZDS 0001, *The Zen Discussion Process*: record obligations, format-record sections.
- ZDS 0002, *zenfmt: Architecture and Implementation*: the superseded AST, builder, transform, and writer sections; everything else preserved.
- ZDS 0003 through 0012: the per-format ledgers gaining facet columns.
- R. Tate, M. Stepp, Z. Tatlock, S. Lerner, "Equality Saturation: A New Approach to Optimization", POPL 2009.
- M. Willsey, C. Nandi, Y. R. Wang, O. Flatt, Z. Tatlock, P. Panchekha, "egg: Fast and Extensible Equality Saturation", POPL 2021.
- M. H. A. Newman, "On Theories with a Combinatorial Definition of 'Equivalence'", Annals of Mathematics 43(2), 1942.
- R. E. Tarjan, "Efficiency of a Good But Not Linear Set Union Algorithm", JACM 22(2), 1975.
- A. V. Aho, S. C. Johnson, "Optimal Code Generation for Expression Trees", JACM 23(3), 1976.
- C. W. Fraser, D. R. Hanson, T. A. Proebsting, "Engineering a Simple, Efficient Code-Generator Generator", ACM LOPLAS 1(3), 1992.
- C. Lattner et al., "MLIR: Scaling Compiler Infrastructure for Domain Specific Computation", CGO 2021, and the MLIR design rationale.
- W3C, *Web Annotation Data Model*, Recommendation, 2017: the stand-off annotation separation facets follow.
- ECMA-376 (Office Open XML), DrawingML: the EMU definition underlying layout-coordinate normalization.
- `zenfmt_advanced`, repository root: the evaluated e-graph proposal, an uncited search-result capture dated 2026-08-07.