

目次

1 はじめに	2
2 最初の変換を最後まで	3
2.1 インストールと基本操作	3
2.2 manifest が証明すること	3
2.3 diagnostic の読み方	3
2.4 終了 code	4
2.5 形式の判定	4
3 1つの共有 document representation	5
4 Reader、Writer、Bundle	6
5 Office と container format	7
6 1つの Writer	8
7 入力を攻撃的な data として扱う	9
8 組み込みと Filter	10
8.1 Zig API	10
8.2 Python API	10
8.3 Browser API	10
9 CLI リファレンス	12
10 ベンチマーク	13
11 Server	14
12 次に読むもの	15

1 はじめに

ドキュメントは、書き手と読み手の間の約束を運ぶものです。ファイル形式は、その間にある歴史的な入れ物にすぎません。zenfmt は Zig で書かれたドキュメント変換ツールで、19 種類の入力形式を読み、GitHub Flavored Markdown を出力します。Pandoc ほど幅広い用途を扱うとは主張していません。小さな engine、明示的な resource limit、再現可能な出力、そして変換時に失われた情報を隠さず伝えることを重視しています。

この文書は zenfmt を使う方のための日本語ガイドです。英語版と同じ章の流れで、CLI、document representation、format reader、Markdown writer、resource limit、Zig と Python API、browser 版、server、benchmark を説明します。code 上の名前、command line option、report code、API field、format 名は、copy や検索ができるよう原文のまま記載します。

設計判断は docs/zds/ の Zen Discussion、略して ZDS に保存されています。ZDS は設計の履歴そのものなので英語のままです。このガイドは実装済みの機能の使い方を説明し、判断の背景が必要な箇所では ZDS へ案内します。

zenfmt には全体を通した 3 つの約束があります。

- 情報の損失は隠さず報告し、terminal と隣接 manifest の両方に残します。
- core は個別形式を知りません。各形式は独立した reader または writer plugin です。
- すべてのドキュメントを信頼できない入力として扱い、archive、stream、nesting、出力 size、memory に明示的な上限を設けます。

2 最初の変換を最後まで

2.1 インストールと基本操作

GitHub Releases から利用する platform の native archive を取得し、展開した実行ファイルをそのまま使えます。この 1 file に CLI と server が含まれ、Java、Python、npm、OCR、VLM、model file は必要ありません。Python から使う場合は `zenfmt wheel` も利用できます。

macOS では repository の Homebrew cask も利用できます。Apple Silicon または Intel に合う同じ self contained archive を GitHub Releases から直接取得します。

```
brew install --cask \
  https://raw.githubusercontent.com/insanai/zenfmt/main/packaging/homebrew/Casks/zenfmt.
rb
```

```
zenfmt report.docx
```

既定では入力の際に `report.md` と `report.md.zenfmt.json` が作られます。前者は Markdown artifact、後者は canonical JSON manifest です。document body だけを stdout へ出す場合は次のようにします。

```
zenfmt report.docx --stdout
```

出力先を明示すると、artifact、manifest、media resource が transaction として公開されます。変換失敗や strict mode による拒否で、不完全な target file は残りません。

```
zenfmt report.docx --output build/report.md
```

現在の binary に含まれる reader と writer は次の command で確認できます。

```
zenfmt --list-formats
```

2.2 manifest が証明すること

manifest は変換の custody record です。source と artifact の名前、判定した format、plugin id、BLAKE3-256 digest に加え、AST schema、document metadata、diagnostic report、plugin preservation data、facet summary が入ります。同じ input、version、option なら同じ byte 列になります。

後の変換で隣接 manifest が見つかり、zenfmt は digest を照合します。一致すれば plugin data を引き継ぎます。一致しなければ stale と報告して無視し、古い metadata を信用しません。

2.3 diagnostic の読み方

diagnostic は Elm compiler を参考にした構成で、毎回 4 つの問いに順番に答えます。

- 何が問題か。
- どこで起きたか。
- zenfmt はそのために何をしたか。
- 利用者は次に何をすればよいか。

各 report には `docx.merged-cells-degraded` のような安定した code があります。automation は、将来改善される可能性のある文章ではなく code を判定に使ってください。loss tier の `degraded` は簡略化して内容を残したこと、`dropped` は内容が出力に存在しないことを表します。

2.4 終了 code

code	意味
0	artifact と manifest を commit しました。warning や情報損失がないという意味ではありません。
1	壊れた container、無効な XML、reader が検出した矛盾などで変換できませんでした。
2	format 名、必要な option、上書き拒否など command の使い方に問題があります。
3	入力が明示された resource limit を超えました。

2.5 形式の判定

拡張子は最初の hint にすぎません。ZIP central directory の part name、OpenDocument と EPUB の `mimetype`、CFB directory stream、PDF header、RTF signature も確認します。拡張子と内容が食い違う場合は内容を優先し、違いを report します。暗号化された document は明示的に拒否し、黙って読み飛ばしません。

3 1つの共有 document representation

すべての reader は同じ document IR を作り、すべての writer はそこから読みます。IR v2 は次の layer で構成されます。

- block と inline の AST が paragraph、heading、list、table、link、image、text などの見える構造を表します。
- attributes が identifier、class、key value attribute を保持します。
- resource store が埋め込み画像などの binary resource を digest で管理します。
- facets が style、tracked revision、page や slide geometry、formula、provenance を sparse な stand off annotation として保持します。
- plugin preservation data が、特定 format にだけ意味があり将来の round trip で必要になり得る情報を保持します。

AST は node ごとの object allocation ではなく、preorder の struct of arrays と `u32` index を使います。subtree は連続範囲なので、走査、copy、limit check が分かりやすくなります。immutable transform は結果を再構築し、変更のない連続 subtree は array range として copy できます。

IR に情報があるからといって、writer がそれを表現できるふりはしません。各 writer は capabilities を宣言し、lowering planner が保持、簡略化、破棄の rule を選んで report を作ります。strict mode は file を書く前にその結果を一貫して判定できます。

4 Reader、Writer、Bundle

core は conversion pipeline を調整しますが、DOCX、PDF、Markdown という固有名を知りません。format plugin は descriptor、reader または writer、capabilities、limits、report catalog を提供します。標準の zenfmt bundle は、配布する plugin を compile time に組み合わせます。

Reader は parse に成功するだけでは不十分です。

- container を展開する前と途中で resource limit を適用します。
- 未対応または簡略化する構造に、実行可能な案内を含む report を出します。
- 構造と index が有効な document IR を作ります。
- media は任意の path に書かず resource store に渡します。
- preservation data は自分の namespace にだけ保存します。

application は必要な format だけの小さな bundle を作れます。Zig project では独自 filter も宣言できます。filter は visitBlock または visitInline を実装し、自分の build.zig で pipeline に compile します。scripting runtime や process 間 serialization は必要ありません。

5 Office と container format

見た目の違う多くの format は、最終的には境界を持つ container として読めます。

- DOCX、XLSX、PPTX、ODT、ODS、ODP、EPUB は通常 ZIP package です。
- 古い DOC、XLS、PPT は CFB compound file を使います。
- RTF は group が nest する token stream です。
- PDF は indirect object、xref、content stream、font mapping の組み合わせです。

ZIP reader は entry 数、1 entry の展開 size、合計展開 size、compression ratio を制限します。仕様上必要な part name だけを読み、archive を disk に展開しません。CFB reader は sector chain、directory tree、stream boundary を検査し、loop を検出します。XML pull parser は nesting、attribute、text size を制限し、external entity を解決しません。

spreadsheet の formula、cell style、merged range は Markdown に完全には表せません。presentation の position、speaker note、animation も同様です。reader は見える text と table structure をできるだけ残し、残りを facet と report で説明します。PDF reader は Zig による native text extraction のみを行い、OCR は実行しません。scan image 自体に抽出可能な text がなければ、その制限をそのまま報告します。

6 1つの Writer

現在の release は Markdown だけを書きます。出力面を狭く保つことで、多くの reader が同じ確認可能な target を共有し、loss policy も理解しやすくなります。

Markdown writer は CommonMark と GFM table を出力します。blank line、改行、escaping を正規化し、file の末尾は 1 newline、trailing spaces はありません。同じ IR は同じ byte 列になります。embedded resource は安定した名前で隣接 media directory に公開され、manifest に記録されます。

直接表せない構造には明示的な lowering rule があります。たとえば merged cell の内容は先頭 cell に残し、covered position は空にします。page break は通常 dropped になります。複雑な style を見た目が同じ Markdown であるかのように扱いません。各判断は report になります。

strict mode で許容する損失を選べます。

```
zenfmt report.docx --strict=content
zenfmt report.docx --strict=structure
zenfmt report.docx --strict=exact
```

content は内容の損失、structure はさらに構造の損失、exact は既知の損失すべてを拒否します。既定の off は変換を完了して損失を報告します。strict refusal は publication より前に起きます。

7 入力を攻撃的な data として扱う

zenfmt は、開発用 machine が偶然処理できた size を安全境界にしません。各 limit には名前、既定値、report code、対処案があります。主な境界は input bytes、archive entry 数、展開 bytes、compression ratio、XML depth、AST node 数、text bytes、resource bytes、output bytes、conversion time です。CLI では安全な範囲内で一部の limit を調整できます。server はさらに request body、concurrency、rate limit、timeout の境界を持ちます。limit 到達は通常の refusal で exit code 3 となり、crash ではありません。

browser profile は tab の memory と実行時間が限られるため、より控えめです。WebAssembly module は filesystem、network、clock、randomness の host import を持ちません。変換は worker で実行され、main page の応答性を保ちます。cancel と timeout は現在の作業を終了し、page を無期限に待たせません。

出所の分からない file では、既定 limit を保ち、manifest と report を確認し、必要な保証に合う strict level を使ってください。1 つの input が成功したことを理由に、外側の service から size や time limit を外さないでください。

8 組み込みと Filter

8.1 Zig API

application は CLI と同じ engine を呼び出せます。

```
const zenfmt = @import("zenfmt");

const result = try zenfmt.convert(gpa, io, .{
    .input = .{ .path = "report.docx" },
    .output = .{ .memory = {} },
});
defer result.deinit(gpa);
```

正確な type と option は利用中の package API を確認してください。caller が allocator と I/O capability を渡し、結果には artifact、manifest、resources、reports が含まれます。binary を小さくしたい場合は `zenfmt_core.Bundle` から必要な plugin だけを組み合わせられます。

Filter は application の compile 済み成果物です。document node を受け取り、再構築した content を返します。走査中に無効になる index を保持せず、builder を迂回して不正な relation を作らないでください。filter による損失も共通 report mechanism で表します。

8.2 Python API

Python distribution は typed で dependency free の wheel で、対応 platform の native bridge を含みます。変換中は GIL を解放します。

```
import zenfmt

conversion = zenfmt.convert("report.docx")
print(conversion.text)
for report in conversion.reports:
    print(report.code, report.problem)
```

`str` は常に path です。memory 上の内容は `bytes` として明示し、名前を渡します。

```
conversion = zenfmt.convert(
    uploaded_bytes,
    name="upload.docx",
    to="markdown",
)
```

`output` を指定すると transaction publication を使い、target の隣に manifest と media を作ります。`zenfmt.Converter` は immutable で再利用できる policy value です。global config、environment、network は読みません。失敗は `ConversionError`、`LimitExceededError`、`UnknownFormatError` などの小さな exception hierarchy で表され、message は CLI と同じく問題、結果、次の手順を説明します。

8.3 Browser API

release の `wasm32-freestanding` bundle は module、ES module adapter、worker、TypeScript declaration を含みます。adapter は artifact ensemble を返し、remote server に fallback しません。static origin から正しい `application/wasm` type で配信し、UI 側にも timeout と size policy を設けてください。

Web application は `npm install @insanai/zenfmt` でも同じ dependency free browser distribution を導入できます。npm package には検査済みの module、adapter、worker、declarations、capability contract が含まれます。native CLI と server に Node や npm は必要ありません。

9 CLI リファレンス

よく使う command は次の通りです。

```
zenfmt INPUT
zenfmt INPUT --output OUTPUT
zenfmt INPUT --stdout
zenfmt INPUT --from docx --to markdown
zenfmt INPUT --strict=structure
zenfmt INPUT --preserve-facets
zenfmt --list-formats
zenfmt --help
zenfmt serve --help
```

stdin には file name という hint がいないため、通常は `--from` が必要です。artifact を stdout に出しても diagnostic は stderr に残り、shell pipeline で別々に扱えます。明示的に許可しない限り、既存出力を上書きしません。

対応する入力 family は次の通りです。

分類	形式
word processing	docx、docm、doc、odt、rtf
spreadsheet	xlsx、xlsm、xlsb、xls、ods、csv、tsv
presentation	pptx、pptm、ppsx、ppsm、ppt、pps、pot、odp
publishing	epub、pdf、html
markup	markdown、asciidoc、rst、text

limit の正確な既定値と diagnostic code 一覧は、英語版 reference chapter、`--help`、machine readable な capabilities を参照してください。program は自然言語ではなく、安定した code と schema version に依存してください。

10 ベンチマーク

benchmark は速度の前に correctness を確認します。corpus manifest は第三者 file ごとの source、format、size、SHA-256 を記録しますが、license の異なる原文書は再配布しません。`benchmarks/fetch_corpus.sh` が同じ byte を取得して digest を検証します。

異なる問いは分けて報告します。

- Native CLI は zenfmt、Docling parser only、AnyDoc、Pandoc の coverage、wall time、CPU time、peak RSS を比較します。
- Browser は WebAssembly の download size、cold ready、warm conversion、native artifact との parity を測ります。
- Server は常駐 zenfmt と Tika Server の startup、warm latency、peak RSS、concurrency ごとの throughput を測ります。
- Output preservation は format ごとに tool neutral oracle を使い、複数の品質判断を 1 score にまとめません。

Docling では OCR、VLM、ASR、layout model、table model、enrichment、accelerator をすべて無効にし、parser backend だけを使います。zenfmt と同じような一般的な machine で実行し、AI pipeline ではなく parser path を比べるためです。未対応 file は unsupported と表示し、無限に遅い値にしたり AI へ切り替えたりしません。

ratio は comparison tool を zenfmt で割った値です。1.0 より大きい場合、この測定では comparison tool がその resource を多く使いました。両者が成功した shared files だけで geometric mean を計算します。release、machine、benchmark lens が違う数字は混ぜません。結果は文脈を示す参考であり、すべての document に対する品質順位ではありません。

11 Server

同じ `zenfmt` 実行ファイルに server、OpenAPI document、database migration、web assets が含まれます。

```
zenfmt serve
curl -s -T report.docx \
  "http://127.0.0.1:8998/api/v1/convert?to=markdown"
```

open mode は既定で `127.0.0.1:8998` のみを listen し、account を持たず stateless です。local tool や管理された sidecar に向いています。`/docs` は組み込み OpenAPI reference、`/openapi.json` は OpenAPI 3.1 contract です。認証なしで web root を開いた場合は、利用できない管理画面ではなく公開 API document へ案内されます。

共有 deployment では secure mode を使います。

```
zenfmt serve --secure --data-dir ./zenfmt-data
```

secure mode は user、role、API key、session、audit log、administration UI を追加します。operator が data directory を明示し、それ以外の場所へ密かに state を作りません。reverse proxy が TLS を担当し、到達可能な bind address を制限してください。API key は一度だけ表示し、log に credential や document body を残さないでください。

conversion endpoint は通常 response と streaming を扱います。client は request size、timeout、retry policy を持ち、HTTP status と structured error envelope で失敗を判断します。health endpoint は生存確認、metrics は運用観測のためのものです。機密 filename のような high cardinality value を metric label にしないでください。

12 次に読むもの

この日本語版は、zenfmt で変換と deployment を行うための実用的な内容をまとめています。英語版には IR layout、format container、diagnostic catalog、limit table、benchmark raw table のさらに詳しい説明があります。設計案と採用しなかった案は英語 ZDS に保存されています。

- project site と browser converter : <https://insanai.github.io/zenfmt/>
- release と self contained archive : <https://github.com/insanai/zenfmt/releases>
- source と issue : <https://github.com/insanai/zenfmt>
- API reference : `zenfmt serve` の起動後に `/docs`

結果が重要な workflow では source、artifact、manifest を一緒に保管し、reports を確認し、必要な保証に合った strict level を選んでください。zenfmt は、確実にできることを明確に実行し、完全に保てないことも同じように明確に伝えることを目指しています。