

The Zen Discussion Process

STATE	INTENDED STATUS
committed	Committed
CREATED	AUTHORS
2026-08-06	Zen Contributors <team@insan.ai>
LAST UPDATED	
2026-08-08	
DISCUSSION	
Process document	
LABELS	
documentation, process	

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

Abstract

1

Introduction

1

Why a converter needs written records

1

Terminology and Scope

1

Discussion Lifecycle

1

Local Workflow Diagram

1

States

1

State Transition Diagram

1

Transition Detail

1

Authoring Rules

1

Format Records

1

Typst Project Layout

1

Build Integration

1

Number Assignment and CI

1

Numbering State Diagram

1

Security Considerations

1

References	1
1. Abstract	20
2. Introduction	21
2.1. Why a converter needs written records	21
3. Terminology and Scope	21
4. Discussion Lifecycle	21
4.1. Local Workflow Diagram	22
5. States	22
5.1. State Transition Diagram	22
5.2. Transition Detail	23
6. Authoring Rules	23
6.1. Format Records	24
7. Typst Project Layout	24
8. Build Integration	24
9. Number Assignment and CI	25
9.1. Numbering State Diagram	26
10. Security Considerations	26
11. References	26
Abstract	1
Introduction	1
Relationship to other records	1
Terminology and Scope	1
Problem Statement	1
Goals and Non-Goals	1
Goals	1
Non-Goals	1
Design Overview	1
The Document AST	1
Shape: a Zig-native document model	1
Storage: flat, preorder, struct-of-arrays	1
Attributes	1
The text pool and side tables	1
Invariants	1
A worked example	1
Why the tree is a tree	1
The Builder	1
Traversal and Rewriting Algorithms	1
Bounded, non-recursive walking	1
Skipping and scanning	1
The rebuild transform, with subtree sharing	1
Complexity summary	1
Staged parsing and streaming rendering	1
Public API and Developer Experience	1
Plugin authoring surface	1
Filters	1
The build.zig analogy, taken literally	1
The filter contract	1
Filters that ship	1
The Plugin Contract	1

Readers	1
Writers	1
The Registry and the Static Router	1
One table	1
Dispatching a runtime pair into a comptime matrix	1
Format taxonomy and roadmap	1
Format detection	1
Adjacent Artifact Manifest	1
Version 1 envelope	1
Loading and carrying plugin data	1
Commit and stream behavior	1
Memory Model	1
Diagnostics and Error Messages	1
The report	1
What it looks like	1
The lossiness tiers	1
Reading the Office Formats	1
The container: OOXML and ODF are ZIP archives	1
The XML layer	1
DOCX	1
Numbering, the hard case	1
Deliberate omissions	1
The other formats	1
The Markdown Writer	1
Repository and Build Layout	1
Command-Line Interface	1
Coding Standard	1
Testing Strategy	1
Security Considerations	1
Operational Considerations	1
Alternatives Considered	1
Decisions Clarified by Review	1
Open Questions	1
The cost of Zig-only filters	1
Image bytes	1
Table alignment source	1
Streaming for very large inputs	1
Filter ordering and conflict	1
Encoding detection	1
Delivery Plan	1
References	1
12. Abstract	20
13. Introduction	21
13.1. Relationship to other records	22
14. Terminology and Scope	22
15. Problem Statement	22
16. Goals and Non-Goals	23
16.1. Goals	23
16.2. Non-Goals	24

17. Design Overview	24
18. The Document AST	25
18.1. Shape: a Zig-native document model	25
18.2. Storage: flat, preorder, struct-of-arrays	27
18.3. Attributes	29
18.4. The text pool and side tables	30
18.5. Invariants	31
18.6. A worked example	32
18.7. Why the tree is a tree	33
18.8. The Builder	33
19. Traversal and Rewriting Algorithms	34
19.1. Bounded, non-recursive walking	34
19.2. Skipping and scanning	34
19.3. The rebuild transform, with subtree sharing	34
19.4. Complexity summary	36
19.5. Staged parsing and streaming rendering	36
20. Public API and Developer Experience	37
20.1. Plugin authoring surface	38
21. Filters	39
21.1. The build.zig analogy, taken literally	39
21.2. The filter contract	40
21.3. Filters that ship	41
22. The Plugin Contract	41
22.1. Readers	41
22.2. Writers	42
23. The Registry and the Static Router	43
23.1. One table	43
23.2. Dispatching a runtime pair into a comptime matrix	44
23.3. Format taxonomy and roadmap	45
23.4. Format detection	45
24. Adjacent Artifact Manifest	45
24.1. Version 1 envelope	45
24.2. Loading and carrying plugin data	46
24.3. Commit and stream behavior	47
25. Memory Model	47
26. Diagnostics and Error Messages	48
26.1. The report	48
26.2. What it looks like	50
26.3. The lossiness tiers	52
27. Reading the Office Formats	53
27.1. The container: OOXML and ODF are ZIP archives	53
27.2. The XML layer	53
27.3. DOCX	53
27.3.1. Numbering, the hard case	59
27.3.2. Deliberate omissions	59
27.4. The other formats	59
28. The Markdown Writer	61
29. Repository and Build Layout	62
30. Command-Line Interface	63

31. Coding Standard	64
32. Testing Strategy	65
33. Security Considerations	66
34. Operational Considerations	67
35. Alternatives Considered	67
36. Decisions Clarified by Review	69
37. Open Questions	69
37.1. The cost of Zig-only filters	69
37.2. Image bytes	69
37.3. Table alignment source	69
37.4. Streaming for very large inputs	69
37.5. Filter ordering and conflict	70
37.6. Encoding detection	70
38. Delivery Plan	70
39. References	71
Abstract	1
Scope	1
Mapping	1
Deliberate omissions	1
Round-trip expectations	1
Security	1
References	1
40. Abstract	20
41. Scope	20
42. Mapping	21
43. Deliberate omissions	23
44. Round-trip expectations	23
45. Security	23
46. References	24
Abstract	1
Mapping	1
Deliberate omissions	1
Round-trip expectations	1
Security	1
References	1
47. Abstract	20
48. Mapping	21
49. Deliberate omissions	22
50. Round-trip expectations	22
51. Security	22
52. References	22
Abstract	1
Mapping	1
Deliberate omissions	1
Round-trip expectations	1
Security	1
References	1
53. Abstract	20
54. Mapping	21

55. Deliberate omissions	22
56. Round-trip expectations	22
57. Security	22
58. References	22
Abstract	1
Mapping	1
Deliberate omissions	1
Round-trip expectations	1
Security	1
References	1
59. Abstract	20
60. Mapping	21
61. Deliberate omissions	22
62. Round-trip expectations	23
63. Security	23
64. References	23
Abstract	1
Mapping	1
Deliberate omissions	1
Round-trip expectations	1
Security	1
References	1
65. Abstract	20
66. Mapping	21
67. Deliberate omissions	22
68. Round-trip expectations	23
69. Security	23
70. References	23
Abstract	1
Mapping	1
Deliberate omissions	1
Round-trip expectations	1
Security	1
References	1
71. Abstract	20
72. Mapping	21
73. Deliberate omissions	23
74. Round-trip expectations	23
75. Security	23
76. References	23
Abstract	1
Mapping	1
Deliberate omissions	1
Round-trip expectations	1
Security	1
References	1
77. Abstract	20
78. Mapping	21
79. Deliberate omissions	22

80. Round-trip expectations	23
81. Security	23
82. References	23
Abstract	1
Scope	1
Mapping	1
Deliberate omissions	1
Round-trip expectations	1
Security Considerations	1
References	1
83. Abstract	20
84. Scope	21
85. Mapping	21
86. Deliberate omissions	21
87. Round-trip expectations	22
88. Security Considerations	22
89. References	22
Abstract	1
Scope	1
Mapping	1
Layout facets	1
Deliberate omissions	1
Round-trip expectations	1
Security	1
References	1
90. Abstract	20
91. Scope	21
92. Mapping	21
92.1. Layout facets	31
93. Deliberate omissions	31
94. Round-trip expectations	36
95. Security	36
96. References	37
Abstract	1
Scope	1
Mapping	1
DOC	1
XLS and XLSB	1
PPT	1
Deliberate omissions	1
Round-trip expectations	1
Security	1
References	1
97. Abstract	20
98. Scope	21
99. Mapping	21
99.1. DOC	21
99.2. XLS and XLSB	22
99.3. PPT	23

100. Deliberate omissions	23
101. Round-trip expectations	26
102. Security	26
103. References	26
Abstract	1
Introduction	1
Relationship to other records	1
The triggering proposal	1
Terminology and Scope	1
A Formal Model of Lossy Conversion	1
Documents and observations	1
The five outcomes of conversion	1
Why lossy alternatives cannot be equalities	1
Problem Statement	1
Goals and Non-Goals	1
Goals	1
Non-Goals	1
Design Overview	1
The Semantic Kernel	1
What stays, and why it stays	1
What changes: one schema, actually one	1
Entities: identity only where identity is needed	1
Sparse Facets	1
The catalog	1
One coordinate system	1
The facet axiom	1
Cost model	1
Extension Nodes	1
Writer Lowering	1
Capabilities, declared at compile time	1
The choice DAG	1
The cost order	1
The planner and its guarantees	1
Strict mode and refusal	1
Normalization, Not Saturation	1
Worked Mappings	1
Core Contract Repairs	1
Limits	1
Security Considerations	1
Operational Considerations	1
Implementation plan: the clean break	1
Acceptance gates	1
Alternatives Considered	1
Open Questions	1
References	1
104. Abstract	20
105. Introduction	21
105.1. Relationship to other records	21
105.2. The triggering proposal	22

106. Terminology and Scope	22
107. A Formal Model of Lossy Conversion	22
107.1. Documents and observations	23
107.2. The five outcomes of conversion	23
107.3. Why lossy alternatives cannot be equalities	24
108. Problem Statement	25
109. Goals and Non-Goals	26
109.1. Goals	26
109.2. Non-Goals	27
110. Design Overview	27
111. The Semantic Kernel	27
111.1. What stays, and why it stays	27
111.2. What changes: one schema, actually one	28
111.3. Entities: identity only where identity is needed	29
112. Sparse Facets	29
112.1. The catalog	29
112.2. One coordinate system	30
112.3. The facet axiom	31
112.4. Cost model	31
113. Extension Nodes	31
114. Writer Lowering	32
114.1. Capabilities, declared at compile time	32
114.2. The choice DAG	32
114.3. The cost order	33
114.4. The planner and its guarantees	33
114.5. Strict mode and refusal	34
115. Normalization, Not Saturation	35
116. Worked Mappings	35
117. Core Contract Repairs	36
118. Limits	37
119. Security Considerations	38
120. Operational Considerations	39
120.1. Implementation plan: the clean break	39
120.2. Acceptance gates	40
121. Alternatives Considered	40
122. Open Questions	41
123. References	42
Abstract	1
Introduction	1
Terminology and Scope	1
Normative implementation contract	1
Problem Statement	1
Goals and Non-Goals	1
Goals	1
Non-Goals	1
Implementation Architecture	1
End-to-end invariants	1
Public Python API	1
Package identity and imports	1

The conversion call	1
Source semantics	1
Destination semantics	1
Reusable policy with Converter	1
Formats and capability discovery	1
Limits and strictness	1
Successful result	1
Manifest model	1
Reports and exceptions	1
Elm-style Python error contract	1
Documentation and compatibility	1
Native Boundary	1
Why a versioned C ABI loaded by ctypes	1
ABI shape	1
Memory artifact ensemble	1
Loading and version checks	1
Concurrency and process behavior	1
Project and Build Implementation	1
Repository layout	1
Division of tool authority	1
Root build steps	1
Repository integration obligations	1
Python configuration	1
Packaging and Release	1
One release version	1
Wheels	1
Source distribution	1
PyPI publication	1
Testing and Quality Gates	1
Python unit tests	1
Native and integration tests	1
Installed-artifact tests	1
Benchmarking the installed wheel	1
Benchmark profiles	1
Measurement rules	1
Correctness before timing	1
Result artifacts and reporting	1
Acceptance gates	1
Security Considerations	1
Threat model	1
Input authority	1
Resource exhaustion and copies	1
Native library loading and supply chain	1
Filesystem output	1
In-process execution	1
Operational Considerations	1
Delivery Plan	1
Phase 1: native contract	1
Phase 2: Python surface	1

Phase 3: packaging and benchmark	1
Phase 4: publication	1
Alternatives Considered	1
Shelling out to the CLI	1
A CPython extension using the limited API	1
cffi	1
A pure-Python reimplementaion	1
A remote service client	1
The uv build backend	1
The repository root as the uv project root	1
Wheel-only publication	1
Returning status instead of raising	1
Mutable global configuration	1
Open Questions	1
Acknowledgements	1
References	1
124. Abstract	20
125. Introduction	21
126. Terminology and Scope	22
126.1. Normative implementation contract	23
127. Problem Statement	23
128. Goals and Non-Goals	24
128.1. Goals	24
128.2. Non-Goals	25
129. Implementation Architecture	25
129.1. End-to-end invariants	26
130. Public Python API	26
130.1. Package identity and imports	26
130.2. The conversion call	27
130.3. Source semantics	28
130.4. Destination semantics	28
130.5. Reusable policy with Converter	29
130.6. Formats and capability discovery	29
130.7. Limits and strictness	30
130.8. Successful result	30
130.9. Manifest model	31
130.10. Reports and exceptions	31
130.11. Elm-style Python error contract	31
130.12. Documentation and compatibility	33
131. Native Boundary	33
131.1. Why a versioned C ABI loaded by ctypes	33
131.2. ABI shape	33
131.3. Memory artifact ensemble	34
131.4. Loading and version checks	34
131.5. Concurrency and process behavior	35
132. Project and Build Implementation	35
132.1. Repository layout	35
132.2. Division of tool authority	36
132.3. Root build steps	36

132.4. Repository integration obligations	37
132.5. Python configuration	37
133. Packaging and Release	38
133.1. One release version	38
133.2. Wheels	38
133.3. Source distribution	39
133.4. PyPI publication	39
134. Testing and Quality Gates	40
134.1. Python unit tests	40
134.2. Native and integration tests	40
134.3. Installed-artifact tests	41
134.4. Benchmarking the installed wheel	41
134.4.1. Benchmark profiles	41
134.4.2. Measurement rules	42
134.4.3. Correctness before timing	43
134.4.4. Result artifacts and reporting	43
134.5. Acceptance gates	44
135. Security Considerations	44
135.1. Threat model	44
135.2. Input authority	44
135.3. Resource exhaustion and copies	44
135.4. Native library loading and supply chain	45
135.5. Filesystem output	45
135.6. In-process execution	45
136. Operational Considerations	45
137. Delivery Plan	46
137.1. Phase 1: native contract	46
137.2. Phase 2: Python surface	46
137.3. Phase 3: packaging and benchmark	46
137.4. Phase 4: publication	47
138. Alternatives Considered	47
138.1. Shelling out to the CLI	47
138.2. A CPython extension using the limited API	47
138.3. cffi	47
138.4. A pure-Python reimplementation	47
138.5. A remote service client	47
138.6. The uv build backend	47
138.7. The repository root as the uv project root	47
138.8. Wheel-only publication	48
138.9. Returning status instead of raising	48
138.10. Mutable global configuration	48
139. Open Questions	48
140. Acknowledgements	48
141. References	48
Abstract	1
Introduction	1
Relationship to existing records	1
Normative language and completion	1
Terminology and Scope	1

Goals and Design Principles	1
Product goals	1
System 1: recognition and action	1
System 2: inspection and understanding	1
Help is part of the interaction	1
Original visual language	1
Design Overview	1
Deliverable contract	1
Zig WebAssembly Implementation	1
Target decision	1
32-bit portability	1
Engine separation required for WASM	1
Low-level ABI	1
Browser profile and memory limits	1
Determinism and artifact parity	1
Public Browser API and Worker Model	1
Distribution shape	1
Ergonomic API	1
State machine	1
Elm-style browser diagnostics	1
Project Site Information Architecture	1
Stable routes	1
Language selection and translated books	1
Homepage content order	1
Download experience	1
Download page wireframe	1
Desktop homepage wireframe	1
Documentation wireframe	1
Mobile wireframe	1
Converter Interaction Specification	1
Input window	1
Output window	1
Progress, status, and focus	1
Responsive behavior	1
Book and ZDS as HTML Documentation	1
One source, two reading modes	1
Sphinx-quality documentation ergonomics	1
Navigation and contextual linking	1
Search	1
PDF publication	1
Benchmark Dashboard	1
What the front page may claim	1
Native lens	1
Browser lens	1
Correctness before timing	1
Result schema and generated presentation	1
Security and Privacy	1
Threat model	1
Capability minimization	1

Content handling	1
Browser isolation and policy	1
Resource exhaustion and recovery	1
Accessibility and Human Interface Gates	1
Build and Repository Integration	1
Proposed repository shape	1
Division of tool authority	1
Root build steps	1
Static assembly	1
Testing and Quality Gates	1
WASM tests	1
Browser coverage	1
Site and documentation tests	1
Performance budgets	1
Release acceptance matrix	1
GitHub Pages and Release 0.2.0	1
Pages workflow	1
Unified release artifacts	1
Operational Considerations	1
Delivery Plan	1
Phase 0: toolchain determinism	1
Phase 1: pure WASM boundary	1
Phase 2: browser API and playground	1
Phase 3: project site and documentation	1
Phase 4: benchmark and release	1
Alternatives Considered	1
WASI in the browser	1
Emscripten or another compiler toolchain	1
Main-thread conversion	1
WebAssembly threads	1
A JavaScript framework and npm build	1
A single-page application	1
Reauthoring the book in Sphinx or Markdown	1
Rendering converted Markdown as HTML	1
Opening output in a browser popup	1
Running competitor benchmarks in every visitor's browser	1
One blended native/WASM headline	1
Third-party analytics and hosted fonts	1
Publishing only the WASM file	1
Known Gaps Outside This Release	1
Open Questions	1
Acknowledgements	1
References	1
142. Abstract	20
143. Introduction	21
143.1. Relationship to existing records	22
143.2. Normative language and completion	22
144. Terminology and Scope	22
145. Goals and Design Principles	23

145.1. Product goals	23
145.2. System 1: recognition and action	24
145.3. System 2: inspection and understanding	24
145.4. Help is part of the interaction	24
145.5. Original visual language	25
146. Design Overview	25
146.1. Deliverable contract	26
147. Zig WebAssembly Implementation	26
147.1. Target decision	26
147.2. 32-bit portability	27
147.3. Engine separation required for WASM	27
147.4. Low-level ABI	27
147.5. Browser profile and memory limits	29
147.6. Determinism and artifact parity	30
148. Public Browser API and Worker Model	31
148.1. Distribution shape	31
148.2. Ergonomic API	31
148.3. State machine	32
148.4. Elm-style browser diagnostics	33
149. Project Site Information Architecture	33
149.1. Stable routes	33
149.2. Language selection and translated books	34
149.3. Homepage content order	35
149.4. Download experience	35
149.5. Download page wireframe	37
149.6. Desktop homepage wireframe	38
149.7. Documentation wireframe	39
149.8. Mobile wireframe	40
150. Converter Interaction Specification	40
150.1. Input window	40
150.2. Output window	40
150.3. Progress, status, and focus	41
150.4. Responsive behavior	41
151. Book and ZDS as HTML Documentation	41
151.1. One source, two reading modes	41
151.2. Sphinx-quality documentation ergonomics	42
151.3. Navigation and contextual linking	43
151.4. Search	43
151.5. PDF publication	43
152. Benchmark Dashboard	43
152.1. What the front page may claim	43
152.2. Native lens	44
152.3. Browser lens	44
152.4. Correctness before timing	44
152.5. Result schema and generated presentation	45
153. Security and Privacy	45
153.1. Threat model	45
153.2. Capability minimization	46
153.3. Content handling	46

153.4. Browser isolation and policy	46
153.5. Resource exhaustion and recovery	47
154. Accessibility and Human Interface Gates	47
155. Build and Repository Integration	48
155.1. Proposed repository shape	48
155.2. Division of tool authority	48
155.3. Root build steps	49
155.4. Static assembly	50
156. Testing and Quality Gates	50
156.1. WASM tests	50
156.2. Browser coverage	51
156.3. Site and documentation tests	51
156.4. Performance budgets	51
156.5. Release acceptance matrix	52
157. GitHub Pages and Release 0.2.0	53
157.1. Pages workflow	53
157.2. Unified release artifacts	53
158. Operational Considerations	54
159. Delivery Plan	54
159.1. Phase 0: toolchain determinism	54
159.2. Phase 1: pure WASM boundary	54
159.3. Phase 2: browser API and playground	55
159.4. Phase 3: project site and documentation	55
159.5. Phase 4: benchmark and release	55
160. Alternatives Considered	55
160.1. WASI in the browser	55
160.2. Emscripten or another compiler toolchain	55
160.3. Main-thread conversion	55
160.4. WebAssembly threads	56
160.5. A JavaScript framework and npm build	56
160.6. A single-page application	56
160.7. Reauthoring the book in Sphinx or Markdown	56
160.8. Rendering converted Markdown as HTML	56
160.9. Opening output in a browser popup	56
160.10. Running competitor benchmarks in every visitor's browser	56
160.11. One blended native/WASM headline	56
160.12. Third-party analytics and hosted fonts	56
160.13. Publishing only the WASM file	56
161. Known Gaps Outside This Release	57
162. Open Questions	57
163. Acknowledgements	57
164. References	57
Abstract	1
Introduction	1
Relationship to existing records	1
Normative language and completion	1
Terminology and Scope	1
Deliverable contract	1
Problem Statement	1

Goals and Non-Goals	1
Goals	1
Non-Goals	1
Design Overview	1
Modes	1
The zencli Library	1
Shape	1
The subcommand rule	1
The zenserve Library	1
HTTP kernel	1
Router and middleware	1
Observability core	1
Authentication core	1
The REST and Streaming API	1
Route table	1
The conversion request	1
Tika compatibility boundary	1
Streaming semantics	1
Server report codes	1
Storage on zaxonlite	1
Dependency and lifecycle	1
Schema	1
Bootstrap	1
Observability Specification	1
Log events	1
Metric catalog	1
The Web Interface	1
The autodoc pattern	1
The ui module	1
The command protocol	1
daisyUI without a framework	1
JavaScript requirement	1
Pages	1
Converter wireframe	1
User management wireframes	1
Session security in the browser	1
The CLI Surface	1
Concurrency and Resource Model	1
connection slots	1
Project and Build Implementation	1
Repository layout	1
Module graph	1
Root build steps	1
Integration obligations	1
Self contained distribution	1
Security Considerations	1
Threat model	1
Input and process containment	1
Credential handling	1

Transport	1
Denial of service	1
Storage	1
Supply chain	1
Testing and Quality Gates	1
Unit	1
End-to-end (loopback)	1
Interface and release gates	1
Benchmark Specification	1
One corpus and one provenance contract	1
Native converter lens with Docling	1
Server lens against Apache Tika Server	1
Correctness before timing	1
Result artifacts and gates	1
Operational Considerations	1
Delivery Plan	1
Phase 1: kernels	1
Phase 2: interface	1
Phase 3: secure mode	1
Phase 4: polish and release	1
Alternatives Considered	1
A third-party Zig HTTP framework	1
An event loop instead of threads	1
A supervised conversion process pool	1
In-process TLS	1
JWTs instead of opaque tokens	1
SQLite directly (or flat files) instead of zaxonlite	1
A configuration file	1
A separate zenfmt-server binary	1
Server-rendered HTML with htmx	1
A SPA framework interface	1
Reusing the ZDS 0015 playground wasm as the interface	1
Compiling the interface wasm per request, as zig std does	1
A live administration event channel	1
Storing conversion history	1
An asynchronous job API for large documents	1
Resolved Questions	1
Acknowledgements	1
References	1
165. Abstract	20
166. Introduction	21
166.1. Relationship to existing records	22
166.2. Normative language and completion	22
167. Terminology and Scope	22
167.1. Deliverable contract	23
168. Problem Statement	24
169. Goals and Non-Goals	24
169.1. Goals	24
169.2. Non-Goals	25

170. Design Overview	25
170.1. Modes	26
171. The zencli Library	27
171.1. Shape	27
171.2. The subcommand rule	27
172. The zenserve Library	27
172.1. HTTP kernel	27
172.2. Router and middleware	28
172.3. Observability core	29
172.4. Authentication core	29
173. The REST and Streaming API	30
173.1. Route table	30
173.2. The conversion request	31
173.3. Tika compatibility boundary	32
173.4. Streaming semantics	32
173.5. Server report codes	32
174. Storage on zaxonlite	34
174.1. Dependency and lifecycle	34
174.2. Schema	34
174.3. Bootstrap	35
175. Observability Specification	36
175.1. Log events	36
175.2. Metric catalog	36
176. The Web Interface	36
176.1. The autodoc pattern	36
176.2. The ui module	37
176.3. The command protocol	38
176.4. daisyUI without a framework	38
176.5. JavaScript requirement	39
176.6. Pages	39
176.7. Converter wireframe	40
176.8. User management wireframes	40
176.9. Session security in the browser	41
177. The CLI Surface	41
178. Concurrency and Resource Model	42
179. connection slots	42
180. Project and Build Implementation	43
180.1. Repository layout	43
180.2. Module graph	43
180.3. Root build steps	44
180.4. Integration obligations	45
180.5. Self contained distribution	45
181. Security Considerations	46
181.1. Threat model	46
181.2. Input and process containment	46
181.3. Credential handling	46
181.4. Transport	46
181.5. Denial of service	47
181.6. Storage	47

181.7. Supply chain	47
182. Testing and Quality Gates	47
182.1. Unit	47
182.2. End-to-end (loopback)	47
182.3. Interface and release gates	48
183. Benchmark Specification	48
183.1. One corpus and one provenance contract	48
183.2. Native converter lens with Docling	48
183.3. Server lens against Apache Tika Server	49
183.4. Correctness before timing	50
183.5. Result artifacts and gates	50
184. Operational Considerations	50
185. Delivery Plan	51
185.1. Phase 1: kernels	51
185.2. Phase 2: interface	51
185.3. Phase 3: secure mode	51
185.4. Phase 4: polish and release	51
186. Alternatives Considered	51
186.1. A third-party Zig HTTP framework	51
186.2. An event loop instead of threads	51
186.3. A supervised conversion process pool	52
186.4. In-process TLS	52
186.5. JWTs instead of opaque tokens	52
186.6. SQLite directly (or flat files) instead of zaxonlite	52
186.7. A configuration file	52
186.8. A separate zenfmt-server binary	52
186.9. Server-rendered HTML with htmx	52
186.10. A SPA framework interface	53
186.11. Reusing the ZDS 0015 playground wasm as the interface	53
186.12. Compiling the interface wasm per request, as zig std does	53
186.13. A live administration event channel	53
186.14. Storing conversion history	53
186.15. An asynchronous job API for large documents	53
187. Resolved Questions	53
188. Acknowledgements	54
189. References	54

1. Abstract

The zenfmt monorepo needs a durable decision-record process for architectural, format, performance, and documentation changes across the conversion library and the zenfmt command-line tool. A document converter accumulates decisions that are invisible in the code and expensive to relearn: which parts of a source format are deliberately dropped, why a construct maps to one Markdown spelling rather than another, and what the plugin contract promises a reader that the engine will never do on its behalf. The project uses Zen Discussions, or ZDS, as RFC/RFD-style Typst documents that support structured reasoning, long-lived references, high-quality PDF output, and an HTML discussion website.

This memo defines the lifecycle of a ZDS, the placeholder numbering workflow, the authoring expectations, and the Typst project layout.

2. Introduction

A Zen Discussion is a Typst document stored in git under `docs/zds/records`. Each discussion is part design memo, part review artifact, and part historical record. The structure is intentionally close to IETF RFCs and Oxide RFDs because those formats force explicit scope, status, rationale, alternatives, and operational considerations instead of relying on implicit context.

ZDS is used for topics such as:

- the document intermediate representation and its compatibility rules
- the plugin contract: what a reader may assume, what a writer must handle
- the mapping from a specific source format to the IR, and the constructs that mapping deliberately discards
- performance and allocation budgets, and the benchmarks that defend them
- security decisions, notably the handling of untrusted archive and markup input
- contributor and documentation process decisions

Active design reasoning, trade-offs, and decisions belong in ZDS. The Typst book under `docs/book/`, when it is written, remains the descriptive manual of the current system; a ZDS records why the system became that way.

2.1. Why a converter needs written records

Every format `zenfmt` reads is a lossy projection onto one shared IR, and every format it writes is a lossy projection back out. The interesting engineering is almost entirely in those two projections, and almost none of it is legible from the code alone. A reader that skips a construct and a reader that has not yet implemented that construct look identical in a diff. Only a record distinguishes them.

This is why a mapping decision — even a small one, such as rendering a single-cell table as a paragraph — belongs in a numbered document with the alternatives that lost.

3. Terminology and Scope

- **ZDS**: a Zen Discussion document
- **placeholder draft**: a local Typst draft using the placeholder number `XXXXX`
- **assigned ZDS**: an accepted discussion with a permanent four-digit number
- **registry**: the Typst metadata list in `docs/zds/registry.typ` used by the index and bundle entry point
- **bundle**: the experimental Typst export target that emits the HTML index, HTML discussion pages, and per-ZDS PDFs from one entry point

This memo defines the repository workflow for ZDS. It does not define future CI automation in full detail, but it does define the expected behavior of that automation.

4. Discussion Lifecycle

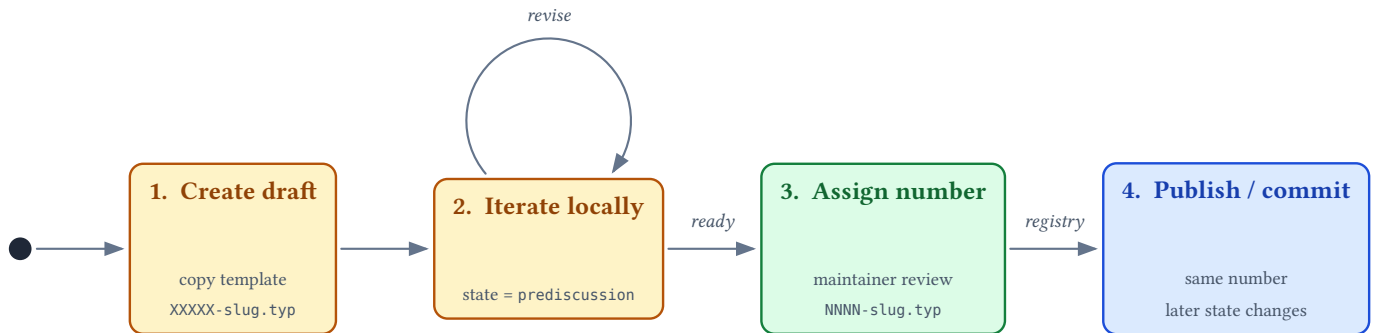
New discussions begin as placeholder drafts.

- Contributors copy `docs/zds/template/rfc-template.typ` to `docs/zds/records/XXXXX-<slug>.typ`.
- The document uses the placeholder number `XXXXX` while the author iterates.

- The registry-driven Zen Discussions index includes placeholder drafts after assigned discussions.
- When the draft is ready, maintainers assign the next permanent ZDS number and rename the file to NNNN-<slug>.typ.
- The registry is updated with the document's area, status, summary, source path, HTML path, and PDF path.

This keeps cloning and local authoring simple. Contributors do not need a global lock or remote numbering check just to start writing.

4.1. Local Workflow Diagram



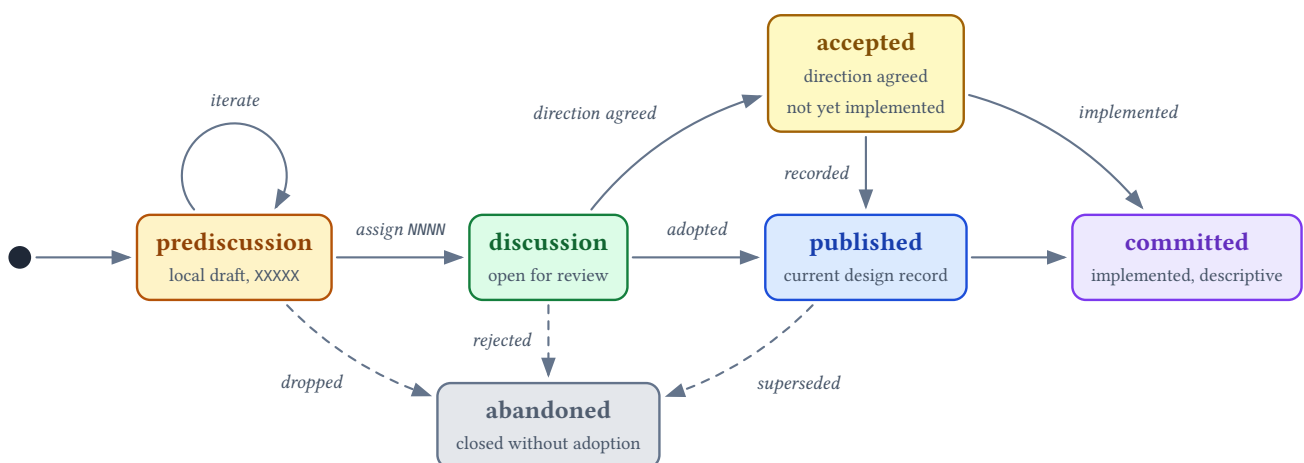
5. States

ZDS documents can move through these states:

- prediscussion: local draft or early working document
- discussion: open for review and feedback
- accepted: accepted as the intended direction, not yet fully implemented
- published: accepted into the repository as the current design record
- committed: fully implemented and now descriptive of the current system
- abandoned: deliberately closed without adoption

5.1. State Transition Diagram

Solid edges are the forward path of a discussion; dashed edges close a document without adoption. The filled dot is the moment a contributor copies the template.



5.2. Transition Detail

Transition	Actor	What happens
start → prediscussion	contributor	Copy <code>template/rfc-template.typ</code> to <code>records/XXXXX-<slug>.typ</code> and iterate locally under the placeholder number.
prediscussion → discussion	maintainer	Run <code>zig build zds-promote --<slug></code> : it assigns the next four-digit number, renames the file to <code>NNNN-<slug>.typ</code> , rewrites the meta-data for discussion, and appends the <code>registry.typ</code> and <code>bundle.typ</code> entries.
discussion → accepted	maintainers	Review converges on the proposed direction; implementation has not landed yet. The registry state changes, nothing is renamed.
discussion → published	maintainers	The document itself is the deliverable (process memos, assessments, records) and is adopted as the current design record.
accepted → published	maintainers	The accepted direction is written up as the repository's current design record while implementation continues.
accepted / published → committed	maintainers	The described system is fully implemented; the ZDS is now descriptive of the current system rather than a proposal.
any pre-committed state → abandoned	maintainers	The discussion is deliberately closed without adoption: dropped while drafting, rejected in review, or superseded by a later ZDS. The number is never reused.

6. Authoring Rules

ZDS documents SHOULD follow the RFC-style template in `docs/zds/template/rfc-template.typ`.

Each ZDS is expected to include, at minimum:

- an abstract or decision summary
- a clear statement of scope and goals
- the main design or proposal
- considerations for security, operations, or workflow where relevant

- alternatives and unresolved questions
- Keep one ZDS per file under docs/zds/records.
- Store document metadata in the `#let zds-*` assignments at the top of the file.
- Update docs/zds/registry.typ whenever a ZDS is added, renumbered, renamed, or changes lifecycle state.
- Build through `zig build` once Zig is installed; Typst remains the document compiler.

The ZDS Typst layer is allowed to use external Typst packages for diagrams, charts, richer tables, or specialized layout when those packages improve the clarity of the PDF output. The repository should prefer existing Typst packages over building a custom extension framework here.

6.1. Format Records

A ZDS that adds or changes a format plugin carries three sections beyond the template, because these are the parts of a converter that reviewers cannot reconstruct from the diff:

- **Mapping table:** every source construct the plugin recognizes, and the IR node it produces. One row per construct.
- **Deliberate omissions:** source constructs the plugin sees and drops on purpose, each with the reason. This is the section that distinguishes a decision from a gap, and it is the section reviewers should read first.
- **Round-trip expectations:** what survives a conversion to the target format and back, and what does not. A converter that claims fidelity it does not have is worse than one that documents its losses.

Two rules follow from the shared IR. A change to `Block`, `Inline`, or their tags affects every plugin at once and requires its own ZDS. A change confined to one plugin’s mapping does not, and may be recorded as an amendment to that plugin’s existing record.

7. Typst Project Layout

The ZDS tree is a first-class Typst project:

- docs/zds/records/ contains standalone ZDS source files.
- docs/zds/template/rfc-template.typ is the starting point for new ZDS drafts.
- docs/zds/registry.typ is the metadata registry used by the index and bundle.
- docs/zds/index.typ renders the PDF/HTML index from the registry.
- docs/zds/bundle.typ uses Typst bundle export to emit index.html, one HTML page per ZDS, and one PDF per ZDS.
- docs/shared/zds.typ and docs/shared/theme.typ provide the shared document frame, styling, and index components.
- docs/book/ and docs/book.typ hold the skeleton of the future zenfmt book. The theme, callouts, and figure helpers are in place; the chapters are not written yet, and no build step compiles them.

Typst bundle export and HTML export are experimental in Typst 0.15. They are useful for the ZDS website, but the repository should still keep standalone PDF compilation for each ZDS as the stable archival path.

8. Build Integration

The root build.zig owns the ZDS build steps. Records are discovered by scanning docs/zds/records, so adding a record never requires a build-file edit:

- `zig build zds` compiles every numbered record to `docs/build/zds-NNNN-<slug>.pdf`.
- `zig build zds -Dzds=<number-or-slug>` compiles a single record; the number may be unpadding (-Dzds=2), and a placeholder draft can be selected by its slug for proofreading before promotion.
- `zig build zds-index` compiles the registry-driven index to `docs/build/zds-index.pdf`.
- `zig build zds-site` compiles the experimental HTML bundle to `docs/build/zds-site/`.

Lifecycle management is owned by `tools/zds.zig`:

- `zig build zds-list` prints registry entries and placeholder drafts, and warns when a record file and the registry disagree.
- `zig build zds-new -- <slug>` creates `records/XXXXX-<slug>.typ` from the template with today's date.
- `zig build zds-promote -- <slug>` performs the prediscussion → discussion transition described above.

Direct Typst commands are useful while editing:

```
typst compile --root docs docs/zds/records/0001-zds-process.typ \
  docs/build/zds-0001-zds-process.pdf
typst compile --features html,bundle --root docs --format bundle \
  docs/zds/bundle.typ docs/build/zds-site
```

9. Number Assignment and CI

The repository supports a local numbering workflow for maintainers who manage discussions directly in git history. The `tools/zds.zig` command, wired into the build as `zds-list`, `zds-new`, and `zds-promote`, automates it:

- list current ZDS entries and placeholder drafts
- assign the next permanent four-digit ZDS number to a placeholder draft
- rename the file from `XXXXX-<slug>.typ` to `NNNN-<slug>.typ`
- rewrite the `zds-number` metadata and transition the document into discussion
- append the metadata entry to `docs/zds/registry.typ` and the export blocks to `docs/zds/bundle.typ`

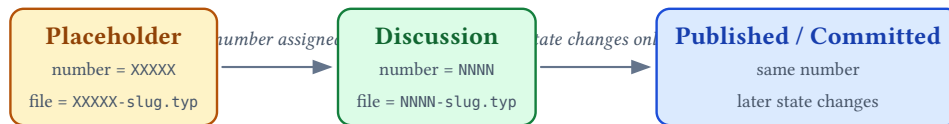
The tool performs ordinary file edits reviewed in the same change as the discussion — there is no hidden state, and every step can still be done by hand. The generated registry summary and area come from the draft's `zds-discussion` and first `zds-labels` entry; review both before committing.

That local flow is sufficient for small teams and direct-maintainer repositories. A future CI workflow can still:

- detect placeholder ZDS files that are ready for discussion
- reserve the next permanent ZDS number
- rename the file from `XXXXX-<slug>.typ` to `NNNN-<slug>.typ`
- rewrite the `zds-number` metadata

Both flows preserve a stable numbered sequence in shared history. Teams can choose local CLI assignment, CI assignment, or a combination where CI validates numbering but maintainers assign numbers intentionally.

9.1. Numbering State Diagram



10. Security Considerations

An ambiguous discussion process creates implementation drift and undocumented design assumptions. For a converter this is not cosmetic. zenfmt parses untrusted input by construction: its whole purpose is to accept a file someone else produced. Decisions about archive expansion limits, entry-count and decompression-ratio caps, XML entity expansion, and path traversal inside container formats are security decisions, and they must stay traceable to an explicit record rather than living as an unexplained constant in a parser.

Requiring explicit sections for scope, considerations, and alternatives reduces the chance that such limits are relaxed later by someone who cannot see why they were chosen.

11. References

- IETF RFCs for explicit memo structure, status, and considerations
- Oxide RFDs for engineering discussion records in a source repository
- Typst bundle export for the multi-file ZDS website and per-ZDS PDFs
- ZDS 0002, the zenfmt architecture and implementation record