Guidedog Discussions GDS 0017

Editable source maps and capability reporting

STATE
discussion
AUTHORS
Vikrant Rathore
UPDATED
2026-11-03
TYPE
Standards Track
CREATED
2026-11-03
DISCUSSION
No external discussion URL assigned
LABELS
library, architecture

Record status

The catalog defines the review process and the meaning of each state. This numbered record forms part of the catalog.

Contents

1. Abstract 1
2. Motivation and scope 1
3. Specification 2
4. Rationale and alternatives 4
5. Security and operational considerations 4
6. Backwards compatibility 5
7. Acceptance criteria and implementation status 5
8. Open questions 5
9. Review history 5
10. References 5

1. Abstract

We give editors four interfaces to Guidedog. An editable source map lists the blocks of one reStructuredText or Markdown document, read with Guidedog's own readers, with the byte spans of their markup and of the text an author edits. Source-preserving operations replace, insert, delete and move blocks, set directive options and add list items as minimal text edits, and refuse any edit whose result does not read back with the intended structure. Preview builds carry output source maps: a `data-gd-source` attribute on each HTML block, and a page map beside each PDF; every build writes a dependency manifest that names the files each page and book was made from. A capability report, from the code the program links, lists formats, builders, directives, roles, extensions, source maps and edit operations; `guidedog capabilities --json` writes it.

2. Motivation and scope

nisaba, a collaborative authoring application, edits Guidedog projects in a visual mode beside the source. Its adapter mapped only headings and paragraphs with code of its own, so it could offer only text replacement, and it kept hand tables of directives that drift from the reader. It rebuilt every PDF of a project for any

change, because no build said which files a book read. These gaps are Guidedog's to close: the readers know the structure, the registries know what exists, and the builders know what they read.
The proposal covers reStructuredText read with the Sphinx layer and Markdown read as MyST, as projects read them. Native Typst documents, embedded diagram and music languages, and configuration files have no block maps. Visual editing of tables, inline markup and the content of diagrams is outside this proposal.

3. Specification

Source maps. The reusable engine crate `guidedoc-edit` provides `load_source_map(text, &MapOptions) -> SourceMap`. `MapOptions` names the format (`Format::Rst` or `Format::Markdown`), the Sphinx layer's configuration and the MyST settings of the project, and a block limit (`DEFAULT_MAX_BLOCKS`, 20,000). The map reads the text with the `sphinx` or `myst` reader, without title promotion and without reading included files, as project builds read a document. It never fails: a document the reader cannot read at all maps to source blocks.
A `SourceMap` holds its format, the text's length, a `truncated` flag, the top-level block ids, and the blocks in document order, a parent before its children. A `Block` holds its `BlockKind`, its `span` from the first byte of its markup to its last non-blank byte, its `content` spans, a title's `level`, its `directive` parts, its `parent` and its `children`. The content is one span per source line, so the editable text is those spans joined by line breaks; it holds no indentation, quote markers, list markers, fences or adornment. A directive's parts are the spans of its name, its arguments, each option (name and value), and its content; MyST options written as a YAML block are marked as such. The Sphinx layer's catalog decides whether a directive takes an argument, so `.. note:: Text` has content on its first line and `.. topic:: Heading` has an argument. Text that no reader node covers, such as `.. highlight::`, becomes a directive or source block, so the top-level blocks cover every non-blank line.

Table 32 Block kinds and their editable content

Kind	Content
`section`	None; its children are its title and body, and its span runs to the next section of its level or a higher one.
`title`	The title text; the span includes the overline, underline or `#` markers.
`paragraph`, `caption`	The text.
`bullet-list`, `enumerated-list`, `definition-list`, `field-list`	None; the children are the items.
`list-item`	The text of its first paragraph, when that paragraph starts on the marker line.
`definition-item`, `field`	The term, or the field name; the children are the body's blocks.
`literal-block`	The code, without its indentation or fences; also a `code-block` directive.
`admonition`, `figure`, `block-quote`	None; the children are their blocks.
`directive`	The content lines when the directive's content is not parsed into blocks.
`table`, `image`, `target`, `substitution-definition`, `transition`	None.
`comment`	The comment text.
`source`	None: any other construct, edited as source.

Operations. `apply_operation(text, &MapOptions, &Operation) -> Result<EditResult, EditError>` applies one operation to the text, whose block ids refer to the map of the same text and options. `EditResult` holds the new text and the edits, each a replacement of bytes `start..end` of the old text, in increasing order and minimal: an edit never rewrites a byte it leaves unchanged. `block_operations(&map, text, id)` lists the operations that apply to one block, and `operations(format)` those a format supports: every operation, in both formats.

Table 33 Operations

Operation	Effect
`replace-text`	Replaces a block's content; continuation lines take the block's indentation and quote markers. An RST title's overline and underline keep their character and take the new

Operation	Effect
	title's display width, unless the adornment was already longer and still covers it. A list item's text is its first paragraph's.
`insert-block`	Writes markup, given at column 0, before or after a block at its indentation, one blank line apart (a quote marker line inside a Markdown quote).
`delete-block`	Removes a block's lines and the blank lines after them, or before them for the last block of its parent.
`move-block`	Deletes a block with its children and inserts its markup next to another block, at that block's indentation.
`set-option`, `remove-option`	Replaces an option's lines, adds an option after the last one or after the arguments, or removes an option's lines; the catalog of the configuration checks the name and the value's type.
`insert-item`	Adds a list item with the anchor's marker; an explicitly numbered RST list is renumbered after it.

New lines take the line break of the file, `\r\n` when its first line ends so.
Every operation reads the edited text again and compares its shape, the depth and kind of every block (and a Markdown title's level), with the shape the operation intends: the same shape for a replacement or an option, the inserted markup's own shape spliced in for an insertion, and so on. A replacement also checks that the block reads back with the new text, and an option edit that the directive has the option with the new value. A difference refuses the edit with `edit.unsafe`, so text that would split a paragraph, close a fence early, or merge a new section with its neighbours never reaches the document. An edit is refused with `edit.truncated` when the map of the text, or of the edited text, lists only its first blocks, because the rest could not be compared. Refusals are `EditError` values with stable codes: `edit.unknown-block`, `edit.unsupported`, `edit.invalid-text`, `edit.unsafe`, `edit.truncated`, `edit.unknown-option` (with "Did you mean"), `edit.option-value`, `edit.missing-option` and `edit.renumber`. `EditError::report` gives an Elm-style report with the block's excerpt.
For example, in `* item one\n  continues` replacing the item's text with `first\nsecond line` gives the single edit `"item one\n  continues"` to `"first\n  second line"`, and inserting the item `new` after `1. one` in `1. one\n\n2. two` gives `1. one\n\n2. new\n\n3. two`.
HTML source positions. `RenderOptions::source_positions` holds a `guidedoc::SourcePositions`: the folder source names are written relative to, and a function that tells how many bytes a source file holds before the text it was read as. With it, the HTML renderer adds `data-gd-source="<source>:<start>-<end>"` to the element of each block: sections, headings, paragraphs, lists and items, terms, literal blocks, figures and captions, admonitions, tables, directives and stand-alone images. A block with several labels writes an empty anchor for each label after the first before its element; the attribute goes on the element, not on those anchors. The span is the block's byte span in the file it was read from, which is an included file for included text. The readers drop a leading byte order mark, so preview builds count its 3 bytes back in: positions are offsets in the file's own bytes, as an editor reads them. Text a reader generates rather than reads from a file, such as `rst_epilog`, autodoc's signatures and MyST substitutions, has a source name in angle brackets and carries no position. Preview builds (`--preview`) set it to the source folder; other builds write no positions, and a preview's pages are never reused for a build or the other way round.
PDF page maps. In a preview, the Typst renderer writes `#metadata("<source>:<start>-<end>")<gd-source>` before each block, by the rules of HTML positions. A block that writes nothing in a book, such as `.. raw:: html`, has no mark, and a block that wraps another of the same span, such as a glossary or an epigraph, adds no second one. The embedded compiler reports where each such element stands (`gd_typst::Compiled::marks`), and the book builder writes `<file>.sourcemap.json` beside the PDF:

```
{"schema": 1, "pdf": "guide-en.pdf", "complete": true, "pages": 12,
 "marks": [{"page": 5, "x": 87.87, "y": 68.03, "source": "intro.rst",
            "start": 11, "end": 17}]}
```

A mark is the block's top left corner in points from the page's top left corner. A block that breaks across pages has one mark, on its first page, and section titles carry no mark of their own; the next block marks

their page. The typst program reports no positions and no page count, so a book it typesets has `"complete": false`, `"pages": null` and no marks.

Dependency manifest. Every HTML, PDF and text build writes `gd-dependencies.json` in its output folder:

```
{"schema": 1, "builder": "pdf", "source_dir": "/work/docs", "shared": ["conf.toml"],
 "navigation": true,
 "outputs": [{"unit": "book:guide.pdf", "files": ["guide-en.pdf", "book-en.typ"],
              "documents": ["index.rst", "intro.rst"],
              "inputs": ["_templates/book.typ", "images/a.svg", "index.rst", "intro.rst"],
              "incomplete": false}]}
```

A unit is a page (its docname) or a book (`book:<file>`); every other unit, such as the files every page shares (`""`), singlehtml's page, the search index and feeds, depends on every document. A unit's inputs are its documents, the files they include, quote or show, the documents its references found their titles, captions and terms in, the project templates it loaded, the files it copied (static files among them), and for a book every file its Typst reads, its template among them. A file a document asked for that does not exist is an input too, so creating it rebuilds the unit; the build reads such a document again until the file appears. `incomplete` marks a unit whose document asked for something that names no file, such as an autodoc object that was not found; it may change whenever a file appears. `navigation` is true because every output also shows each document's title, section titles and place in the toctree, so a change to those changes every output. Paths are relative to the source folder, or absolute outside it; `shared` names what every output depends on. The output manifest records the files a unit read (version 3), so a unit kept from an earlier build keeps its inputs.

Capability report. The internal crate `gd-capabilities` provides `load_capabilities(&gd_plugins::Registry, Option<&gd_conf::Config>) -> Capabilities` and `write_json`. Every entry comes from linked code: readers and renderers from `guidedoc_defaults::registry()`, formats with their suffixes, block kinds and operations from `guidedoc-edit`, builders from `gd_env::BuilderKind::ALL`, directives and roles from the Sphinx layer's new catalog (`guidedoc_sphinx::directives`, `roles` and `find_directive`, over Docutils' tables, which `guidedoc-rst` now exports), and extensions from `gd_conf`. Each directive and role says where it comes from (`docutils`, `sphinx`, `myst`, `project` or `domain:<name>`) and how it becomes available: `default`, `config` (with the extension that enables it), or `plugin`. With a project's configuration, `enabled` says whether that project has it. The report's `schema` is 1; new fields may appear without a change, and a changed or removed field changes it. `guidedog capabilities [PROJECT] [--json]` prints a summary or the JSON form.

4. Rationale and alternatives

We map with the readers instead of a separate parser, so the map cannot disagree with the page. The readers give exact spans for blocks but not for every piece of markup inside them; we find directive parts and adornments in the text the reader marked, with the reader's own directive rules. Reading the result of every edit again costs two reads per edit, which is small for one document, and it turns every subtle markup interaction into a refusal instead of a corrupt document. Content as one span per line, rather than one span with indentation inside it, lets an editor show and edit text without knowing the markup's indentation rules. For HTML we considered a separate map file keyed by element ids; an attribute needs no ids and survives templates and plugins that rewrite the page. For PDF we considered Typst's own spans of the generated source; labelled metadata is stable across Typst versions and is queried through the public introspector. A dependency manifest beside the output, rather than a query interface, works for a caller that only reads files. The plugin interface of this version adds builds, publishers and commands, not markup, so no directive is `plugin`-enabled yet; the value exists so the report need not change when plugins can add markup.

5. Security and operational considerations

Maps read untrusted text with the readers' limits and read no files; a map lists at most its block limit and says when it is truncated. Operations are pure functions of their text and options. Source positions name files relative to the source folder and appear only in preview builds; the page map and dependency manifest name absolute paths only for inputs outside the source folder. The metadata a preview book carries is invisible and does not change its layout.

6. Backwards compatibility

Existing builds are unchanged except for the new `gd-dependencies.json` file in their output. The output manifest's version is now 3, so the first build after an upgrade writes every unit again. `RenderOptions` gains `source_positions`, `guidedoc` gains `SourcePositions`, `PositionCache` and `relative_source`, `gd_typst::Options` gains `marks`, `Compiled` gains `marks`, and `UnitRecord` and `OutputRecord` gain `files_read`; callers that build these with struct literals must add them, which the 1.0 beta versioning permits. `guidedoc-rst` exports `standard_directives`, `find_standard_directive`, `standard_roles` and `column_width`; nothing existing changes.

7. Acceptance criteria and implementation status

Implemented and published in Guidedog 1.0.0-beta.11:

- Map tests for every block kind in both formats, for directive parts, gaps, the innermost block at an offset, configuration-dependent directive rules, and the JSON form.
- Operation tests in both formats for each operation, including Unicode title widths, quoted Markdown text, MyST field and YAML options, renumbering, and every refusal code.
- A corpus test that maps every RST and Markdown file of Guidedog's documentation, checks that blocks nest and cover every non-blank line, and checks that replacing a block's text with the same text changes no byte.
- Catalog tests for directive lookup, declarations and requirements; capability report and command tests.
- End-to-end tests: a preview page carries the exact span of a paragraph; a preview book's page map places a paragraph of each document; the dependency manifest lists includes, quoted files and images per page, every chapter per book, and survives an incremental build.

Not implemented: block maps for native Typst and the embedded languages; visual operations on tables and inline markup; positions for section titles in PDF page maps; page maps from the typst program.

8. Open questions

Whether nisaba needs marks for every PDF line rather than every block is open until its viewer uses the page map. Whether insertion should choose the separation of a compact list or definition list by its neighbours, as list items do, is open.

9. Review history

On 3 November 2026 the maintainer asked for these interfaces so that nisaba can offer visual operations beyond text replacement, map its previews back to source, rebuild only affected PDFs, and report capabilities from Guidedog's registries; this record proposes them and records the implementation.

Before the release, three independent reviews of the implementation found faults, which we fixed with tests that fail without the fix: maps that panicked or mislabelled RST titles, Markdown quotes and unclosed fences; edits that a truncated map, a changed field name, a YAML comment or CRLF line endings let through; positions three bytes short after a byte order mark, on generated text or on a label's anchor; and manifests that hid a page named `dependencies`, left out what singlehtml, the search index and static files were made from, or listed markers and refused templates as files. Fixing the manifest also showed that Guidedog kept a document's page when a file it included was created later; it now reads such a document again.

10. References

- NSD 0002, nisaba architecture, gates A-02, A-15 and A-17; NSD 0004, the Guidedog engine adapter.
- GDS 0003, Guidedoc conversion core: the document model and spans.
- GDS 0011, Sphinx built-in extensions; GDS 0013, static publishing (preview builds).
- Typst 0.15 documentation: `metadata`, labels and `locate`.

Lifecycle event

```
2026-11-03: prediscussion -> discussion. Assigned a permanent number and
opened for discussion.
```