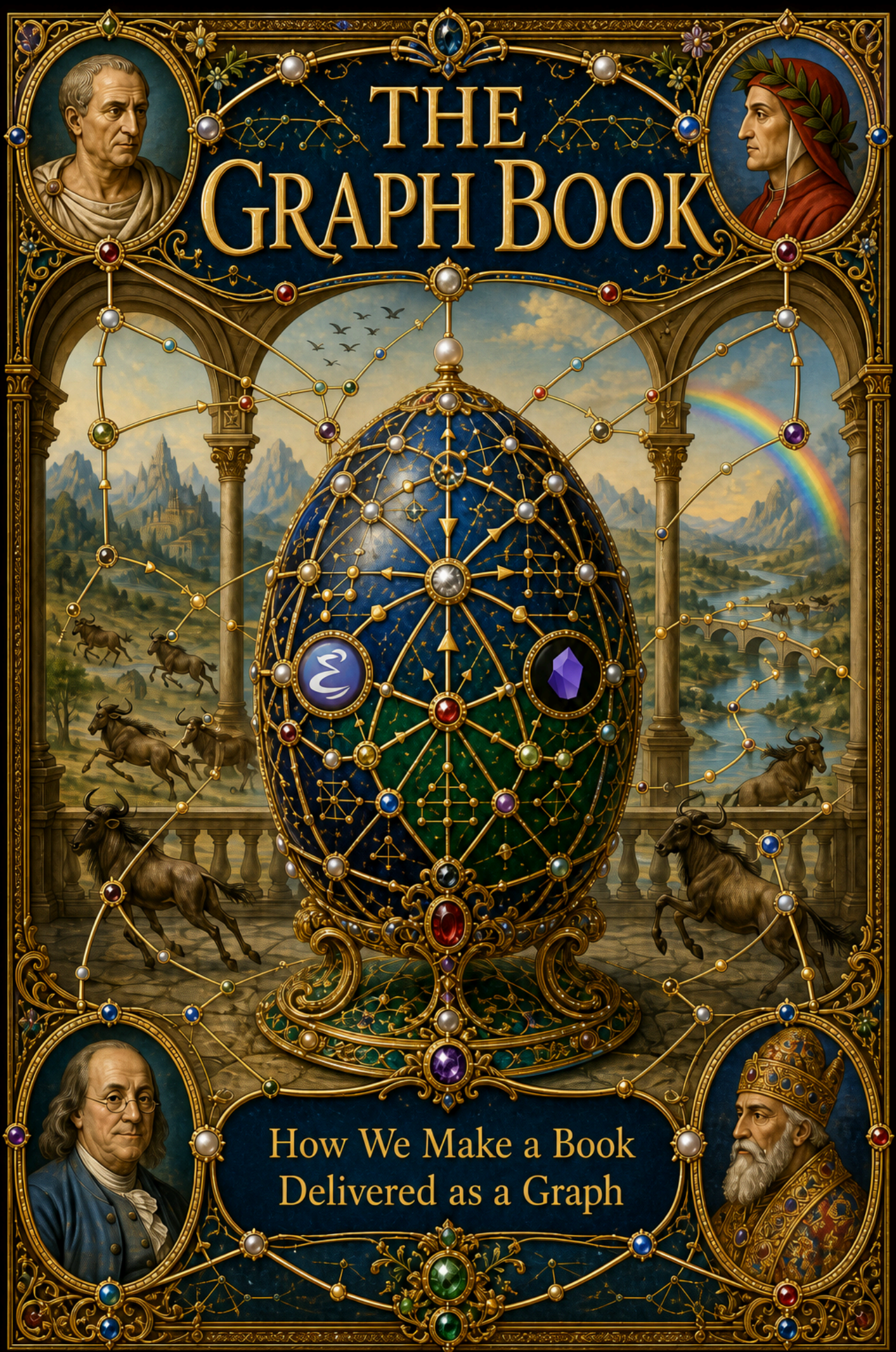


THE GRAPH BOOK

How We Make a Book
Delivered as a Graph



The Graph Book

A Revolutionary Way to Organize a Knowledge Graph as a Book

Alexy Khrabrov

2026-09-03

Contents

Preface	2
Why a Book Should Be a Graph	3
The Shape of a Graph Book	4
Profiles	4
Products	4
The reader index and the units ledger	4
The Reader	5
Identity	5
The Sail Rust Book: A Codebase as a Book	6
Fragments	6
What this taught us	6
Lighthouse Republics: Voices, Editions, and Anthologies	7
The graph of a history	7
The triptych	7
Rosetta: Comparative Voice Engineering	8
Cicero: Quotations and Their Witnesses	9
Records, not footnotes	9
The first dictionary	9
Dante: Every Word and Every Translation	11
Alignment as a declared strategy	11
Every word	11
The reader's controls	12
Dictionaries That Help Each Other	13
The first graph: forms to lemmas	13
The second graph: through English	13
The lesson	14
The Two Readers	15
Obsidian	15
Emacs	15
Publishing a Graph Book	20
Making Your Own	21
What Comes Next	22
Acknowledgements	23

Preface

This is a book about books that are graphs, and about a press that learned, over four titles and two years, how to make them.

The claim in the subtitle is not modest, so let me say plainly what it means. A knowledge graph is a set of things and the relations among them: a chapter and the source file it explains; a sentence and the archival letter it paraphrases; a quotation and the three translations that render it; a word and the dictionary entry that parses it. Everyone who has tried to organise knowledge this way has run into the same wall. Graphs are wonderful to build and dreadful to *read*. They have no beginning, no argument, no voice; the reader is handed a map and asked to make the journey alone.

A book is the opposite kind of object. It has an order, an author, a place to start and a place to stop. It is the most successful interface for knowledge ever devised, and its weakness is exactly the graph's strength: everything a book knows about the world outside its pages is flattened into a footnote, a citation, a "see also".

The Graph Book is our answer to both. You organise the knowledge as a graph — honestly, completely, with every edge recorded and every node bound to a source — and you deliver the graph *as a book*: a spine of ordered pages that reads from first to last, in which every claim can be opened to the evidence beneath it, every quotation to its witnesses, every foreign word to its dictionary, without leaving the page. The reader who wants a book gets a book. The reader who wants the graph finds it one tap away, under every sentence.

We did not arrive at this by theory. We arrived at it by making four very different books and noticing that they wanted the same machinery. The *Sail Rust Book* wanted its chapters to open the code they describe — twenty thousand excerpts of a living codebase. *Lighthouse Republics*, a history of the American republic's Venetian inheritance, wanted its twenty-nine editions and nine hundred anthology documents laid side by side so an editor could hear the voices and choose among them. *Cicero on Himself* wanted its Latin quotations to open their witnesses, in Latin, English, and Russian, aligned. And Dante's *Commedia* wanted nine translations in two languages beside the Italian, tercet by tercet, and a dictionary that could explain every one of the poem's twelve thousand word forms — including the ones Dante spelled his own way seven centuries ago.

Each of those is a graph. Each is delivered as a book: a PDF you can print, an EPUB for the reading app, a hosted web edition, an Obsidian vault that runs on a phone, and an Emacs Info bundle that runs on anything with a keyboard, including an iPhone under a terminal emulator. The chapters that follow explain how, title by title, and then generalise: what a graph book is made of, how the two readers work, how one is published, and how you can make your own.

Why a Book Should Be a Graph

Consider three ordinary readers.

The first is learning Rust by reading a book about a large Rust project. Every few pages the book says “the planner does this” or “this is implemented in the session module”. She wants to see it. Not a snippet the author copied into the book a year ago, but the file, at the commit the book describes, with the lines in question highlighted, and the freedom to scroll up and read the rest.

The second is an editor of a long history. The book has been through twenty-nine releases in four voices, and every release has a slightly different anthology of documents behind it. He wants to put three versions of one paragraph beside each other, see how far they diverge and where, and record his preference without touching the canonical text.

The third is reading Dante in Italian with school Italian. She wants Longfellow beside the tercet, then Cary when Longfellow is too tidy, then the Russian she grew up with; and when she meets *ch'i'* or *eterno* or *avea* she wants the dictionary to say what they are and *why* they are spelled that way, not to shrug.

None of these readers wants a knowledge graph. All of them want a book that knows more than it shows. The insight of the graph book is that these are the same thing seen from two sides. The book is the spine; the graph is everything the spine can reach. The spine gives the graph an order and a voice; the graph gives the spine its depth.

Three consequences follow, and they organise everything else in this book.

The spine is the source of truth for order. A graph has no first node. A graph book has one ordered reader index — a list of pages with stable identities, bound to a version of the manuscript and a commit of the source — and every reader, on every device, walks that same list. Nothing in the delivered graph is allowed to invent a reading order from backlinks or filenames.

Every edge is a record, never a link. A citation in a graph book is not a footnote saying “Atticus 4.8a”. It is a record: an identifier, a canonical text reference, the Latin, the English, the Russian, the source each came from, the character span in that source, the translator, the year. Records can be validated, counted, and refused. A reader who opens the record sees the witnesses; a validator that checks the record can tell whether the witnesses still say what the record claims.

The same graph feeds every reader. A graph book is compiled, not authored, into each of its forms. The PDF, the vault, and the Emacs bundle are three projections of one set of chapter files, record maps, and dictionaries. When Dante gained Alexander Ilyushin’s complete 2008 Russian translation, a parser and one registry entry placed it beside Lozinsky in both private study readers; the same entry’s rights field kept it out of both public readers. No one edited three editions, and no one maintained a separate blacklist.

The rest is engineering, and the engineering is the point. What follows is how each of the four titles forced a piece of it into being.

The Shape of a Graph Book

Before the titles, the anatomy they share. A graph book at First Pair Press has one description, `vault.build.json`, in the repository that owns the manuscript. It names the slug and title, the **profile**, the ordered **reader** pages, the **evidence** the pages can open, and the **products** to deliver, and optionally an **emacs** block for the Info edition. Everything the publisher does starts from this file.

Profiles

A profile is what kind of knowledge the graph holds, and therefore what kinds of evidence a page may cite. There are three:

Profile	The book is about	Evidence kinds	Collections
code	a codebase	code file, code range, example, diagram, document	code tree, examples, documents
history	an argument from sources	source passage, anthology entry, claim, visual, document	source corpus, anthology, visuals
trptych	aligned witnesses of one work	parallel passage, editorial, annotation, visual, document	trptych corpus, editorials, annotations

The Sail Rust Book is a code book; Cicero is a history book; Lighthouse Republics and Dante are triptych books, though as we will see they are triptychs of very different things — editions of one prose work in one case, translations of one poem in the other.

A profile is also a promise about rights. Evidence a title may not redistribute is carried as a metadata-only record — identity, public URL, hashes, the claims it supports — never as bytes. The validator for each profile refuses a vault that copies what its profile says it may only describe.

Products

A product is a delivery shape. `desktop` is the complete editorial vault: every unit, every record, every witness, as large as the graph is. `mobile` is a compact Reader product derived directly from the same source for a phone — with file and byte ceilings, exact cited sources, compact illustrations, and a full static Markdown fallback — never produced by deleting files from the desktop vault. `preview` is a bounded public sample, often the front matter and first movement, with the complete source-navigation corpus behind it. A title declares whichever products it needs; the products are projections, and the contract forbids treating one as a reduced form of another.

The Emacs bundle is a fourth kind of product, projected from the same reader order and evidence: a book manual, a references manual that opens below the text, an offline lexicon, and the Texinfo source of both.

The reader index and the units ledger

Two files make a vault a graph book rather than a folder of notes. The reader index (`_data/reader.json`, or `_data/parallel-reader.json` for aligned editions) is the spine: page identities, titles, paths, parts, in reading order, bound to the manuscript version and source commit. The units ledger

(`_data/units.jsonl`) is the graph’s node table: one row per unit of the book — a paragraph, a code fragment, a sense unit, a tercet — with its identity and its place. Ledgers are generated from the source; they are “auditable without parsing rendered prose notes”, and they are never a second source of truth.

The Reader

Obsidian reads Markdown; it does not, by itself, know that a note is page 12 of a book, that a fenced block is a code fragment, or that a JSON file is a chapter of aligned tercets. The FirstPair Reader plugin knows. It takes the reader index and shows the book in one dedicated view with one navigation rail — **Previous · Up · Back · Top · TOC · Next** — whose *Back* is a bounded, Reader-internal history that never drops the reader out into an unrelated note. Every page keeps a complete static Markdown fallback, so a vault without the plugin is still a book.

Emacs reads Info. The FirstPair Emacs reader knows that the Info manual is a book, that the second manual below it holds the references, that the words underlined in the text are in the lexicon, and that the regions of an aligned page belong to particular translations. We will come back to both readers in a chapter of their own.

Identity

Every delivered artefact carries the commit it was built from, and the publisher refuses an artefact whose commit is not the pushed HEAD of the source repository. This sounds bureaucratic and is the single most useful rule in the system: it means that a vault on a phone, a bundle in an Emacs on a train, and the PDF on a shelf can be traced to the same line of the same file, and that a reader’s annotation made against a unit identity survives the next release.

The Sail Rust Book: A Codebase as a Book

The first graph book was a book about code, and it set the pattern by being uncompromising about one thing: the code in the book had to be the code.

Sail is a large Rust project — a Spark-compatible engine built on Arrow and DataFusion. The *Sail Rust Book* is a twenty-one-page guide to it: a reader’s guide to how the book itself is built, then twenty chapters from the architecture down to the current codebase edition. A conventional technical book would quote the code. This one *cites* it.

Fragments

The builder walks a checkout of Sail at a pinned commit and makes one note per source file — 2,027 of them at the edition validated in July 2026. It then extracts *fragments*: every Rust `fn`, `struct`, `enum`, `trait`, `impl`, `mod`, `type`, `const`, and `static`; every Python `class` and `def`; every Markdown heading — 20,201 fragments, each with a file, a start line, an end line, and the revision. A fragment is a node. The chapters are nodes. The edges are chosen: for each chapter the builder selects the fragments that chapter explains and appends a “Generated Code Fragment Index” — a sequence of fenced `sail-fragment` cards, one per excerpt.

The reader sees a card with the symbol, the file, and the line range. The bundled plugin turns the card into a button, **Open code fragment**, which opens the code-file note beside the chapter with the excerpted lines highlighted — the whole file, not a snippet, at the commit the book describes.

What this taught us

Three things, all of which survived into every later title.

First, *the source repository owns the graph*. Sail’s vault is generated by the title’s own `build-obsidian-vault.py` and checked by its own `check-obsidian-vault.py`; FirstPair supplies the layered guide, the transactional candidate directory, the privacy checks, and the differential comparison against the previous vault. The framework never guesses what a chapter cites; the title says.

Second, *units need a ledger*. Sail’s `_data/units.jsonl` — “the publishing compatibility ledger used by FirstPair” — emits units of kind `chapter`, `code-file`, and `code-fragment`. Every later profile has a ledger of its own kinds, and the publisher’s structural check of a vault is, at bottom, “does it have a `Home.md` and a units ledger”.

Third, *size is a design constraint, not an afterthought*. Twenty-seven thousand files and a hundred and thirty megabytes is a fine desktop vault and an impossible phone vault. The ceilings on Sail’s product declaration (`maxFiles`, `maxBytes`) became the mechanism by which every later title declares what a device can carry, and the reason the mobile product exists as a separate projection.

The Emacs edition of the code profile is designed and documented — code excerpts become reference nodes, and `C-c C-f` opens the delivered file itself in the lower window — and awaits its bundle; the Sail Rust Book is named as the title that will prove it.

Lighthouse Republics: Voices, Editions, and Anthologies

The second graph book was a history, and it broke the assumption that a book has one text.

Lighthouse Republics is a history of Venice’s inheritance in the American republic. It has been through twenty-nine releases, and the releases are not merely versions: they are *voices*. There are four release families. **Main** (ten releases) is the canonical line. **South** (six) is documentary and periodic, derived from the historian Assis Brasil’s manner. **East** (six) is quieter and more reflective. **West** (seven) is spare and direct. Each family reads the same sources and tells the same story in a different register, and each release of each family has its own anthology of documents — 878 edition-specific anthology documents in all, because the wording and even the selection of what to quote changed as the voices developed.

The graph of a history

The full editorial vault is the largest graph we have delivered: 50,819 sense units, 49,012 direct transformation events, 2,947 conceptual lineages, more than 117,000 files, and 3.4 million wikilinks — a vault so dense that Obsidian needs an eight-gigabyte heap to index it. Its evidence graph is four ledgers in `_data/`: `editions.jsonl`, `units.jsonl`, `events.jsonl`, `lineages.jsonl`. A *unit* is a sense unit of one edition. An *event* is a transformation of one unit into another — this sentence of Main 2.0 became that sentence of Main 3.0. A *lineage* is the chain of such events for one idea across every release. Given the lineages, the vault can answer a question no printed history can: show me how this paragraph was said in every voice, and where it came from.

Each release’s Anthology/ folder holds the documentary texts at that release’s pinned commit — *A5 – Adams on Venice as Constitutional Evidence* and its hundreds of siblings — with links to the people, places, sources, and chapters that cite them. Separately, Sources/ holds one note per public URL, research object, or figure, recording its locator, every unit and anthology occurrence, the deterministic mapping into anthologies, and the provenance of any binary. Anthology and source are two layers on purpose: the anthology is what the edition chose to quote; the source is what exists.

The triptych

The Reader-visible instrument of this graph is the triptych: a set of panes showing the same passage in different editions. The book declares four comparison sets — the *Edition 4 decision* (four panes across the editions that competed to become the canon), *Main 2.0 / 3.0 / 3.1*, *South / East / West final*, and a pair, *South 2.2.1 first versus final*. Alignment across editions is structural, by lineage, with manual literary decisions recorded in an overrides file rather than imposed on the text. The Review view lets an editor record a preference pane by pane and “never silently composes a manuscript” — a rule we came to think of as the ethics of the triptych: the graph may make comparison easy, but it may not make a choice on the author’s behalf.

The mobile product of this vault — 29 editions, 3,988 Reader pages, 50,819 sense units, 8,609 triptych slices, 214 source notes, in 905 files and 55 MiB — is the proof that a graph this large can be carried in a pocket if the projection is designed rather than trimmed.

Rosetta: Comparative Voice Engineering

Lighthouse Republics needed a format for comparing witnesses, and the format outgrew the book. Rosetta is that format: a general comparative editorial method for aligned editions of one work, implemented in two profiles.

The first profile is the translation Rosetta of *The Invented Enemy*, whose rail shows **F / R / C** — the Fable draft, the Codex review, and the Codex text. The second is the edition-and-voice Rosetta of Lighthouse Republics, the “Venice Rosetta”, whose rail shows **S / E / W** — the final South, East, and West editions — and which extends to pairs and to four panes. The word *trptych* names a three-pane set; *Rosetta* names the method.

Rosetta’s contribution to the graph book is that it made comparison *measurable without making it comparable across kinds*. Each aligned fragment has a durable key of the form {set-id}/{lineage-id}; each pane carries a divergence score and a feature vector (the NMSRECF vector of the Venice profile) and shows difference marks — · | || ||| — so the eye can find where the voices part company before the mind reads why. And the scores of one profile are never to be compared with those of another: an 80 among editions of a history is not an 80 among translations of a novel. Page numbers are deliberately excluded from durable review targets, because pages are an artefact of one rendering and the graph outlives every rendering.

Dante’s readers inherit all of this without knowing it. When a column header says *Cary (1814) ≈*, the mark is Rosetta’s honesty about alignment carried into a triptych of translations: the row is comparable, not equivalent.

Cicero: Quotations and Their Witnesses

The third graph book is a memoir of Cicero in the first person, and it is where the graph learned to speak more than one language.

Cicero on Himself is built on quotations. Its narrator is Cicero; its authority is Cicero's own letters, speeches, and treatises; and every quotation in the text is an edge to a witness. The history profile exists for this: its evidence kinds are the source passage, the anthology entry, the claim, the visual, and the document, and its Reader opens a quotation to the passage it comes from.

Records, not footnotes

The edges are recorded in a bilingual quote map — thirty-three passages in the published preview — and each record is the model of what a graph-book citation is:

```
{"id": "quote-att-4-8a", "work_title": "Letters to Atticus",  
  "citation": "4.8a", "urn": "urn:cts:latinLit:phi0474.phi057:4.8a",  
  "latin": "Ubi nihil erit, quod scribas, id ipsum scribito.",  
  "english": "When you have nothing to write, write and say so.",  
  "latin_source_id": "pg58418", "latin_span": "8903-8908",  
  "english_span": "8940-8945", "translator": "E. O. Winstedt",  
  "publication_year": "1912"}
```

The identifier is stable. The canonical reference is a CTS URN, the standard scholarly address of a passage of Latin literature. The Latin and the English are both quoted, each with the source file it was taken from and the character span within it, so a validator can reopen the source and confirm the quotation still stands there. A second map, of the same thirty-three passages, carries their Russian readings, merged by identifier — the first time a First Pair graph held three languages of one edge.

Around the quotations stands the rest of the history graph: paragraph units, source and claim records, eleven research dossiers as document evidence, thirty-three illustrations with visual provenance, and the bilingual Latin/English corpus of sources — Project Gutenberg texts, the Latin Library, the Internet Archive — each with its download log and derived-file table. The complete vault is a research archive; the mobile product keeps all twenty-four Reader pages, their thirty-three cited passages, and reader-sized forms of the illustrations; the preview holds the complete Proem with the full source-navigation corpus behind it.

The first dictionary

Cicero's Latin is the reason the Emacs edition exists, and the reason every later graph book has a dictionary.

A reader of Cicero in Latin needs a Latin dictionary that understands inflection: not *scribito* but “future active imperative, second person singular, of *scribo*”. FirstPair's lexicon layer for Latin is built on Whitaker's WORDS, the classic morphological analyser, and it *projects*: it takes every Latin word that actually appears in the book, analyses it, and delivers a table of forms, stems, endings, entries, and glosses that answers offline. When the Russian translation of the quotations arrived, the lexicon learned to answer in a second language — English from WORDS, Russian from the Russian Wiktionary and a reviewed supplement, covering 398 of the 414 Latin forms in the preview — and the reader gained a selector, **E / R / E+R**, that chooses which languages the dictionary speaks.

That selector, the projected lexicon, the references manual opening below the text, and the marked words a reader can step through with C-c C-n are the Emacs edition. It is an Info manual written

directly by the builder — not by `makeinfo`, so that every marked word's line and column stay exact — and it was the form in which Cicero first ran on an iPhone.

Dante: Every Word and Every Translation

The fourth graph book is a poem, and it is where the machinery became general.

Dante’s *Commedia* is one hundred cantos of terza rima, 14,233 lines, 4,811 tercets. The graph book of it has the Italian as its spine and, beside every tercet, any of nine translations: in English, Longfellow’s line-for-line verse of 1867, Cary’s blank verse of 1814, Norton’s prose of 1891–92, and Sibbald’s terza rima Inferno of 1884; in Russian, Dmitry Min’s complete translation in Dante’s own metre (Inferno 1855, Purgatorio 1902, Paradiso 1904), Vasily Petrov’s terza rima Inferno (1887), A. P. Fedorov’s Inferno (1898), V. V. Chuiko’s prose Inferno (1894), and Dmitry Minaev’s Purgatorio and Paradiso (1874–79). All nine are public domain; the English come from Project Gutenberg, the Russian from Lib.ru’s transcriptions of the nineteenth-century editions, offered both in their pre-1918 spelling and modernised by rule.

Alignment as a declared strategy

A translation is registered once, with its language, its maker, and — the decisive field — its *alignment*:

- **line**: the translator kept Dante’s line count (Longfellow, Sibbald, Min, Petrov, Fedorov). Tercet *i* takes lines $3i..3i+2$; every row shows the same verses.
- **proportional**: verse of another length (Cary; Minaev, who runs to two and a half times Dante in places). The lines are cut at the same fractions of the canto as the tercets.
- **prose**: Norton and Chuiko are cut into sentences, each placed by its share of the canto’s characters.

Only the first asserts that a row shows the same verses. The other two are approximate and are marked $\approx$ wherever they appear. The registry also records *coverage* — Sibbald finished only the Inferno, Minaev only the Purgatorio and Paradiso — so the readers offer a translation only on the parts it covers, and a transcription that lost a page (Lib.ru’s Min lacks twenty-nine lines of Inferno XVI) leaves the gap blank rather than shifting the alignment.

The chapter files are the same `firstpair-aligned-chapter-v1` JSON the Obsidian Reader and the Emacs builder both consume: a title and a list of units, each with its source lines and the lines of every translation that covers it. The Ilyushin parser read the complete 2008 edition into one hundred cantos and 14,233 lines, corrected three structural OCR defects against the page images, and then passed an exact canto-by-canto line-count comparison with the Italian. One registry entry put **Лозинский · Илюшин** next to each other in the Russian rotation. Because the entry is marked study-only, the private vault and bundle gained it while the public editions remained byte-clean. This is what “adding a translation” means in a graph book: parse, align, declare, validate; the projections follow.

Every word

The dictionary is the deepest node of the Dante graph and the part we are proudest of. The poem has 12,766 distinct word forms. The Italian lexicon layer — the same design as Cicero’s Latin layer, over the English Wiktionary’s Italian entries instead of Whitaker — analyses 11,921 of them (93.4%, and 98.9% of the running text) to a modern headword, and says *how*: “apocope of *amore*”, “old form of *diceva*”, “elision of *il*”. The restoration rules are the poem’s: elisions and apocope, the old verb endings, enclitic pronouns with doubled consonants, spelling alternations like *eterno*, prothetic *i-*, paragoge, and the Wiktionary’s own “form of” links, followed even when written only as gloss text.

Senses come from the English Wiktionary and FreeDict for English and, for Russian, from the Russian Wiktionary, the Italian Wiktionary’s translation tables inverted, a pivot through English senses,

FreeDict, and a small reviewed supplement, with a second pass through related lemmas so that *gaetta* reaches *gaietto* and *gaietto* reaches Russian. Russian covers 11,412 forms. Every entry carries the poem's own lines as examples; a form the edition cannot analyse says so. Nothing is guessed.

One defect of this pipeline deserves recording, because it is general: the Wiktionary's conjugation tables name each verb's auxiliary as a form row, and until those rows were skipped, *avere* and *essere* each resolved to eighteen thousand lemmas. Graphs built from other graphs inherit their mistakes at scale.

The reader's controls

Because the Dante graph has several translations per language, its readers grew the controls that a triptych of translations needs. In Obsidian, each translation column's header names the translation and rotates through the language's translations on a tap; + opens a second column of the same language, so Longfellow and Cary can stand beside one tercet; a dropdown in the toolbar does the same where headers are hidden. On a phone held upright the tercet stacks — Italian first, translations indented beneath — and returns to columns when the phone is turned. The dictionary opens over the empty column, or as a band across the bottom the reader can drag to size, and can stay open, keeping its last entry, while pages turn.

In Emacs, C-c C-v rotates the translation under the cursor and C-c C-b shows or hides a second translation of that language. In a terminal the top menu exposes **Tr-next** and **Tr-prev** directly, acting on the primary or second slot under point; **2nd** returns from two editions to one in a single action, however many alternatives exist. A persistent line immediately below the menu names the visible editions in the order their text appears — **Longfellow · Лозинский · Илюшин**, for example — while the bottom message names the edition just selected and remains until another command replaces it. State is therefore visible without changing it.

The published edition is one library entry with two *versions* — Italian and English; Italian, English, and Russian — each a complete set of book, vault, and Emacs bundle. That, too, was machinery Dante forced into being: a catalog entry that carries language editions rather than a slug per language.

Dictionaries That Help Each Other

The Dante dictionary is worth a chapter of its own, because it is the clearest case we have of graphs supporting one another — of a graph that could not answer a question borrowing the answer from a neighbour.

The question was simple: what does this Italian word mean in Russian? The obvious source, an Italian–Russian dictionary, was FreeDict’s *ita-rus*, and looked up naïvely it answered for 13 per cent of Dante’s word forms. English fared little better at 18 per cent. The reason was not the dictionaries’ size but their shape: they are keyed by headword, and Dante does not write headwords. He writes *avea* for *aveva*, *amor* for *amore*, *ch’i’* for *che io*, *eterno* for *eterno*, *gaetta* for *gaietta*. A dictionary keyed by the modern lemma is a graph with no edge from the poem’s words to its own nodes.

The first graph: forms to lemmas

The first repair was the analyser: a graph from every surface form in the poem to its modern headword, built from the English Wiktionary’s Italian entries — their inflection tables give the forms, their “form of” links give the historical and dialectal spellings — and from a set of restoration rules for what the Wiktionary does not know. With it, 93.4 per cent of the poem’s forms reach a headword, and the analyser says how they got there. English coverage followed at once, because the same Wiktionary entries carry English senses: every analysed form has an English answer.

Russian did not follow at once, because no single Russian source is keyed the way the analyser keys. So the Russian dictionary was assembled as a *union of graphs*, each contributing the edges it has:

- The **Russian Wiktionary’s Italian entries** — Italian headword to Russian senses, direct, but sparse.
- The **Italian Wiktionary’s translation tables** — each Italian entry lists its Russian equivalents; a direct edge again, from a different community with different coverage.
- The **Russian Wiktionary’s translation tables, inverted** — a Russian entry lists its Italian translations; read backwards, that is an Italian-to-Russian edge nobody wrote as such.
- **FreeDict ita-rus** and a small **reviewed supplement** of the poem’s own vocabulary.

Merged by folded headword, these four took Russian from 13 per cent of forms to the high eighties. Each is a graph; none was made for Dante; together they cover most of him.

The second graph: through English

The remaining gap closed through a fifth graph, and this is the step that deserves the name of the chapter. The English Wiktionary’s Italian entries have English *senses*, and its English entries have Russian *translations* keyed by sense. So an Italian headword with no Russian edge of its own can reach Russian by way of the English words that define it: *gaietto* → “gay, merry” → *весёлый*. This is a pivot through the English sense graph, and it is dangerous if done carelessly — English words are ambiguous, and a pivot through *bank* will happily hand you a riverbank for a savings account. It is done at the level of the sense, not the word: the pivot follows the English gloss the Italian entry actually gives, into the Russian translations the English entry gives for that gloss, and it is labelled in the result (“via English”) so the reader knows which edges were walked.

A second pass extended the pivot along the analyser’s own edges. When a form’s headword has no Russian and no usable English gloss, the pass follows the headword’s *related* lemmas — the form-of links, the “Dantesque form of *gaietto*” written only as gloss text — and then the words of the headword’s

gloss, until something answers. That is how *gaetta* (Inferno I, the leopard's spotted hide) reaches Russian: the poem's form to the Wiktionary's *gaetta* entry, its gloss text to *gaietto*, *gaietto*'s English sense to the English entry, the English entry's translation table to Russian, with the chain recorded.

The result is 11,412 forms with a Russian answer — 95.7 per cent — of which 6,799 are marked as derived, so a reader can tell a direct lexicographic edge from a walk across three dictionaries. The 509 forms still without Russian are mostly proper names, which want transliteration rather than translation, and a few hundred rare words that a human supplement will catch over time.

The lesson

Nothing here is specific to Italian or Russian. It is a pattern: when the graph you need does not exist, find the graphs that do, make their node identities agree (here, the folded headword and the analysed form), and walk across them — recording every crossing, so the answer carries its own provenance. The same machinery, pointed at the German or Spanish Wiktionary, gives Dante a German or Spanish dictionary; pointed at Whitaker's WORDS and the Russian Wiktionary, it gave Cicero his. The dictionaries help each other exactly because they are graphs, and the book that delivers them can show the reader the path.

One more thing the union taught us: a graph inherits the errors of its parts at scale. Every conjugation table in the English Wiktionary names the verb's auxiliary as if it were a form of the verb, and until those rows were skipped, *avere* and *essere* each had eighteen thousand lemmas. Provenance is what let us find it.

The Two Readers

A graph book is read in two very different environments, and the difference is instructive.

Obsidian

Obsidian is a Markdown editor with a graph view, which makes it the natural home for a knowledge graph and an unnatural home for a book. The FirstPair Reader plugin closes the gap. It is small (a few hundred lines), offline (no network, no subscriptions to vault events), ships disabled and is enabled by the reader, and is exercised in a real DOM by a test that opens a fixture vault, renders a chapter, taps a word into a sharded dictionary, switches layouts, follows Home links, and survives a vault that is still syncing.

Its contract is the one described above — a single ordered Reader index, one rail, bounded Back, static fallback — plus what aligned editions need: the `parallel-reader.json` index of languages, translations, dictionaries, and pages; the three layouts (columns, stacked, and automatic by pane width and phone orientation); the dictionary drawer that sizes itself to the column it should cover, measured from the grid rather than the window; dictionaries delivered as prefix-keyed shards under four megabytes, because Obsidian Sync’s per-file ceiling and a phone’s patience are both finite; and a memory of where the reader stopped — page, scroll position, languages, translations, the open entry — restored on the next opening.

Two lessons from the phone deserve to be written down. Obsidian’s leaf content is a CSS containing block, so a drawer positioned in window coordinates lands in the wrong place; the drawer must live in a frame of its own inside the view. And a plugin whose files Sync has replaced keeps running the old code until it is toggled off and on; every delivery note must say so.

Emacs

Emacs reads Info, the hypertext of the GNU manuals, and Info is older than the web. It is also exactly right for a graph book: nodes with names, cross references between manuals, a directory, and a reader that runs in a terminal. FirstPair’s bundle is two Info manuals — the book and its references — an offline lexicon in tab-separated tables, the Texinfo source of both, the reader’s Lisp, and a manifest binding all of it to a commit. `install.sh` registers the manuals with `install-info`; a standalone `firstpair-reader` package carries the reader for people who install it once and read many bundles.

The reader’s keys are the graph’s edges. `C-c C-d` opens the word at point in the dictionary window; `C-c C-n` and `C-c C-p` step between marked words; `C-c C-r` opens the references; `C-c C-f`, in a code edition, opens the delivered file; `C-c C-t` chooses the languages; `C-c C-v` and `C-c C-b` rotate and pair translations; `C-c C-g` opens the glossary. On a phone, where chords are a burden, the same commands are single letters — `d`, `t`, `v`, `b`, `,`, `.`, `r`, `?` — and compact controls in the book and dictionary mode lines. In an aligned text, a tap on *any* source-language word, not only a pre-marked one, passes the surface form straight to the lexicon and updates the existing Dict pane without moving focus or opening keyboard input. The poem and dictionary remain read-only: the tap is a graph query, not an editing gesture. The reader remembers where it was left, per bundle, and returns there. In a terminal it also turns on `xterm-mouse-mode` itself: touch is part of the delivered reader, not a setup exercise left to the reader.

Reading Dante on a phone in Emacs

The Emacs reader's most demanding home is an iPhone running Alpine Linux under iSH, an x86 emulator in an app. It taught the reader nine things. Four changed the Reader; five crossed the boundary into the terminal app itself.

First, that a phone wants **taps, not chords, and state you can see**. Reader 1.46 divides the Emacs controls into two non-redundant bars. The middle bar owns the book: **Dict · ▲ · ▼ · ◀c · c▶ · Top · Refs · ?**. The lowest bar owns words and translations: **Tr< · Tr> · 2nd · Lang** at the left, with << · < · > · >> flush right. The four word targets are twice their original width. The single chevrons move one source word; the double chevrons move to the next or previous *significant* word, skipping a configurable set of Italian articles, prepositions, pronouns, and common forms of *essere* and *avere*. All four open the dictionary as they move. They walk the Italian of an aligned canto, never the translation lines.

Translation choice also became visible state rather than a cycle hidden behind one button. The terminal menu has direct **Tr-Eng** and **Tr-Rus** submenus. **None** hides that language; every edition below it is a checkbox, so selecting an unchecked translation adds it beside those already showing and selecting a checked one removes it. A header row names the visible translations in their actual display order. Its controls remove an edition, move it earlier within its language, or move a whole language block first. **Tr<** and **Tr>** replace only the translation under the cursor. **2nd** adds the configured favorites, Palma and Ilyushin in the Dante edition, without disturbing any other visible text; the inverse command keeps only those favorites. The state underneath is small: one ordered list of editions per language and one ordered list of visible languages. A tap on any Italian word looks it up immediately, and the current dictionary word remains underlined in the poem. The poem and dictionary stay read-only; a tap is a graph query, not an editing gesture.

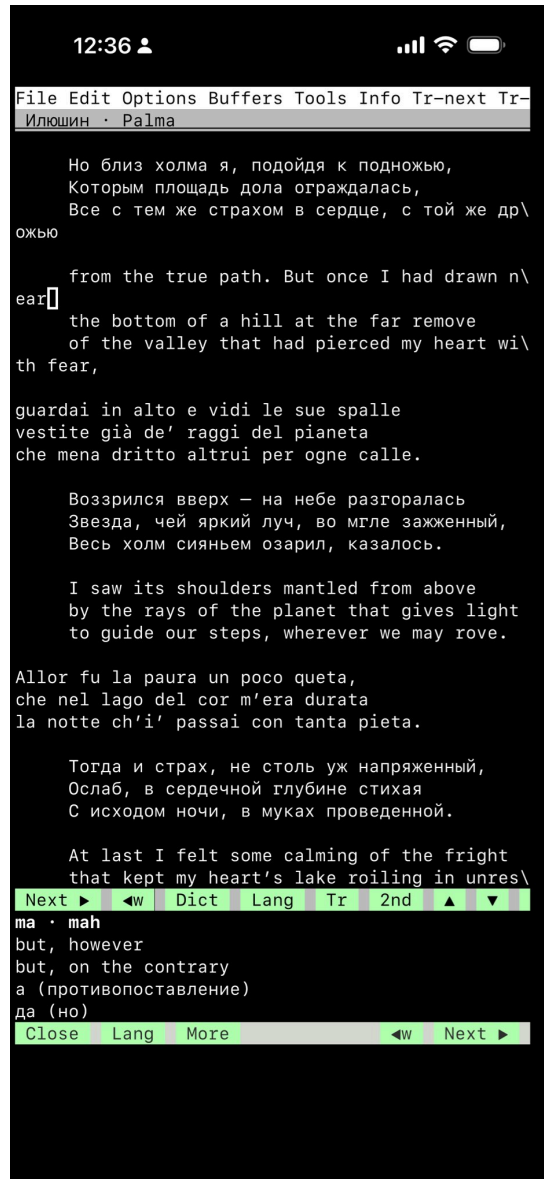


Figure 1: Dante’s Italian, English and Russian translations, and compact dictionary in terminal Emacs on an iPhone during development of the touch Reader.

Second, that **nothing large may load before the first page**. Registering a bundle used to parse its whole region table — 4.8 megabytes, 51,552 rows for the study Dante — and the first lookup parsed a 30-megabyte gloss table and a 2-megabyte forms table; a Mac does this in a second, iSH in minutes. Regions now ship grouped by node with a byte-offset index, so a canto reads only its own; forms and glosses ship as first-letter shards, so a lookup reads one slice. Registration fell from 210 milliseconds to one. The first dictionary lookup can still be perceptible, so Reader 1.45 cycles Loading dictionary., Loading dictionary..., and Loading dictionary... while it parses, then leaves Dictionary ready: WORD in the echo line. Later cached lookups are quiet.

Third, that a reader on a phone is **resumed, not opened**. The reader saves its state — node, point, languages, translation choices, second translations — to a small file on every page turn, after a few idle seconds, and at exit, and firstpair-read returns there; the Obsidian Reader does the same per

edition. And because a bundle ships its own Lisp for readers without the package, the bundle's `init.el` must defer to an installed package when one exists and use its own only in batch — the day it did not, a phone ran last week's reader while reporting this week's version.

Fourth, that **an update must be less work than reading the update notice**. The Dante bundle carries `dante.sh`. It fetches the tiny `firstpair-reader.tar.sha256` release record first. If its version is already installed, it downloads nothing else and opens the book. If the version is new, it downloads the tar, verifies the SHA-256, verifies that the archive really is `firstpair-reader` at the declared version, installs the new package before removing older descriptors, and refuses to replace files underneath a live Emacs. It then starts a fresh terminal Emacs with the intended Dante bundle passed explicitly. The same script can be run every day; doing nothing is one of its tested outcomes. If the network is absent, failure to fetch the release record or package falls back to the installed Reader and local book; malformed metadata and wrong hashes remain fatal, because offline reading is a convenience and accepting unverified code is not.

Fifth, that a full-screen touch reader requires the terminal to understand **when a tap is not a request to type**. Hiding iSH's keyboard gave Info the whole iPhone screen, but an ordinary tap asked the native terminal to become first responder and brought the keyboard straight back. No Emacs command could fully repair that boundary, because iOS focus belonged to the host app. We therefore added a persisted **Keep Keyboard Hidden** switch to iSH itself. When a terminal program has enabled VT mouse reporting and the switch is on, a one-finger tap remains an application mouse event but the native focus request is suppressed. Outside mouse mode, tap-to-type behaves normally; in mouse mode, a two-finger tap explicitly restores the keyboard. The result is not Emacs squeezed into a terminal. It is Info as full-screen iPhone hypertext: links, words, translation controls, the dictionary, and all four wide word targets remain tappable while the keyboard stays away until invited.

Sixth, that **a tap must arrive at the cell the finger touched**. WebKit also synthesizes mouse events for a touch, and a terminal that trusts a misplaced synthetic coordinate can turn every menu choice into the first item. That failure first looked like a Reader defect: choosing a translation selected **None**. The decisive control was stock Emacs: choosing **File** → **Quit** opened **Find File**. Emacs's terminal menu also requests DECSET 1003 pointer movement before release, a mode `hterm 1.91` ignored. 1Unix now uses the live `UITouch`, reports movement at its cell, and completes a tap at the original touch-down cell before finger drift can change the target. Named Reader commands make menus serializable; correct pointer state makes them selectable.

Seventh, that **the most-used controls can belong to the host without surrendering their meaning to it**. 1Unix now carries a persistent native Reader strip below the terminal: `Tr< · Tr> · 2nd · Lang · << · < · > · >>`. It is not an input accessory, so it remains on screen when the software keyboard is hidden; when the keyboard returns, the strip moves above it. Each button sends the Reader's ordinary single-key command, bypassing terminal mouse decoding while leaving Emacs in sole charge of translation and word state. A hold on `Tr<` or `Tr>` performs the opposite direction; a hold on `2nd` keeps only the favorite translations; a hold on `Lang` cycles language state backward. The strip deliberately excludes page, canto, dictionary, and reference controls already present in Emacs. The ordinary keyboard accessory likewise keeps four explicit arrow keys and removes its redundant gesture pad.

Eighth, that **a finger must be able to drag the page without ceasing to be a precise pointer**. Under DECSET 1003, movement shorter than three quarters of one rendered line remains a tap at its

original cell. A deliberate vertical drag becomes natural-direction wheel steps delivered at that same cell, so Emacs scrolls the poem or Dictionary pane in which the gesture began. Reader 1.43 binds mouse-4 and mouse-5 locally across text, mode lines, buttons, and the spaces between them, because terminal Emacs cannot be assumed to provide a global wheel command. It scrolls one line at a time and restores the previously selected window. A scroll must never look up the word beneath the finger, follow a link, resize a pane, reveal the keyboard, or move the other pane.

Ninth, that **optional native furniture is a layout state, not merely a hidden view**. **Show Dante Reader Bar** is a separate persisted 1Unix setting, enabled by default and independent of **Keep Keyboard Hidden**. Turning it off gives its height back to the terminal immediately; turning it on restores the same controls without restarting Emacs. The first implementation could leave the terminal black because it deactivated a weak storyboard constraint that UIKit then released. The repaired design keeps one strong terminal-to-bar constraint active forever and hides the strip by collapsing its height to zero. After layout, 1Unix asks hterm to resize, invalidate, redraw, and synchronize its cursor. Automated tests repeat the transition and verify terminal text and reclaimed height; the signed-phone acceptance protocol also guards both cold-launch states with the keyboard hidden and visible.

The generic keyboard-focus change lives in the alexy/ish fork as feature/keep-keyboard-hidden-in-mouse-mode and was proposed upstream as iSH pull request 2796. The personal build, **1Unix**, carries that change plus the Reader strip, scrolling contract, fonts, name, icon, bundle identifier, and signing profile on linux/ui, so it can coexist with stock iSH. Branding and book-specific controls are deliberately absent from the upstream patch. **Keep Keyboard Hidden** defaults off; the personal Reader bar defaults on. Both are choices, because changing focus and occupying screen space are policies even when they make the terminal a better book.

The Reader release is produced by one command from the canonical Lisp. It builds `firstpair-reader-<version>.tar`, copies it to the stable `firstpair.org/emacs/firstpair-reader.tar` route, writes the version-and-hash sidecar, and fails its tests if the source headers, package metadata, tar, or sidecar disagree. The launcher is tested for current-version skip, verified installation, and offline fallback before deployment. The app and Reader now add gesture, menu, state-machine, and signed-device tests to that chain. This is automated Emacs delivery in the graph-book sense: source, release record, installer, bundle, host app, and public bytes are nodes whose identities must agree.

Two older lessons remain true. The Info format forbids periods, commas, and colons in node names, and Emacs truncates a labelled cross reference at the first period, so citations use hyphenated node names, and the builder writes the Info bytes itself so that the line and column of every marked word are known exactly. And a table that a laptop parses in a second must be assumed to take minutes somewhere else.

Publishing a Graph Book

A graph book is published by a pipeline with gates, and the gates are where most of the value lives. In order:

1. **A contract.** `FIRSTPAIR.md` at the source repository's root names the slug, the shelf, and the default edition, and records the build and publish commands. The source repository owns the manuscript, the graph, the validators, and the guide fragments; the press owns the builder, the readers, the catalog, and the site.
2. **A book package.** The unified builder takes `book.build.json` and produces PDF (Typst), EPUB, single-file HTML, and chapter HTML, with a `VERSION.md` naming the edition, the version, and the source commit. The builder verifies its own toolchain against a lock file before it starts.
3. **Vaults and bundles.** The vault builder projects the products; the source-owned validator checks each one — Dante's counts cantos, units, and dictionary shards, refuses a translation whose rights are not public domain in a public edition, and refuses Cyrillic in the English one. The Emacs builder writes the bundle and `firstpair-emacs validate` runs the inventory, structural, link, lexicon, `makeinfo`, and `emacs --batch` gates. The shared Reader has its own release gate: one version across all Lisp files, a package tar whose metadata agrees, and a SHA-256 sidecar published before the launcher is allowed to install it.
4. **Guides.** Every vault and bundle ships a complete first-use guide composed from four layers — the shared handbook, the profile, the product, and the title's own notes — so a reader who has never opened Obsidian or Emacs is not sent chasing links.
5. **Identity.** Both repositories must be clean and pushed; the bundle manifest must name the pushed HEAD. Docs are written before the last build, never after, or the build binds to a commit that no longer exists.
6. **A dry run, then the live run.** The publisher stages the package, uploads it to Blob storage one title at a time, updates the catalog and the route maps, writes the title's README, copies versioned files to iCloud, runs the catalog check, builds and smoke-tests the site, deploys, and verifies that the live catalog and hosted readers match the local entry. With `--emacs`, the same transaction selects the product declared by the title, validates its bound source commit, archives one clean root folder, renders its composed guide, uploads both, records their hashes, and creates the stable download routes; there is no separate hand upload for an Emacs edition.
7. **Rights.** A full edition is published only for text the source repository is licensed to distribute. Restricted witnesses — Lozinsky's Russian Dante, a modern English translation the owner bought — live in a separately named study edition that the validator will not pass as public.

Two gates protect the vaults specifically. Obsidian must be fully quit before any vault directory is regenerated or archived, because it rewrites workspace and index files in the background; and a vault delivered through Obsidian Sync must have the originating Mac's Sync toggles — all file types, community plugins — switched on, because they are off for a newly connected vault and the failure looks like a broken plugin rather than a missing file.

Making Your Own

Everything described here is open, and designed to be copied.

Choose a profile. If the book is about a codebase, the code profile and the Sail Rust Book’s builder are the model: one note per file, fragments by symbol, cards in the chapters. If it argues from sources, the history profile and Cicero’s quote map: records with canonical references, source spans, and every language of the quotation. If it aligns witnesses of one work — editions, drafts, translations — the triptych profile and Dante’s registry: parsers that return `{(part, chapter): lines}`, an alignment strategy per witness, and coverage.

Write the description. `vault.build.json` with the reader pages, the evidence, the products, and, if you want the Info edition, the emacs block with its direntry, parts, and lexicon. The schema is published and validated.

Build, validate, compare. `firstpair-vault plan | build | compare | snapshot` generate a candidate, check it, and compare it with the previous vault; the existing vault stays authoritative until the candidate passes comparison and a human reads it in the Reader.

Add a language. For a triptych of translations, a language needs a dictionary from the source language into it in the shared `firstpair-reader-dictionary-v1` shape. FreeDict publishes such dictionaries under Creative Commons for dozens of pairs; the Wiktionary extractions pinned by FirstPair supply four kinds of glossary — entries, inverted translation tables, direct translation tables, and a pivot through English senses — and the projection joins them to every analysed form of the text. Dante’s `docs/ADDING-TRANSLATIONS.md` walks the whole path from finding a public-domain text to seeing it in the column header.

Publish. The seven steps above, recorded in your `FIRSTPAIR.md`, and the press’s playbook in its `AGENTS.md` for whoever — or whatever — runs them.

What Comes Next

The graph book is young, and the next steps are visible from here.

The code profile wants its Emacs bundle, with the Sail Rust Book as the proving title, so that a reader in a terminal can open a chapter's fragment in the file it came from. The history profile wants Lighthouse Republics' voices in Info, with the triptych rendered as regions the reader rotates the way Dante's translations rotate. Dante's private study graph now has Ilyushin beside Lozinsky; the public edition wants a new language — German, Spanish, French, and Polish all have public-domain *Commedias* on Wikisource — and with it a fourth dictionary language.

And the readers want each other. A graph book read in Obsidian on a phone and in Emacs on a laptop is one book; the place a reader stops in one should be the place the other opens. The identities are already shared. The rest is a small file.

The larger claim stands. A book can be a graph without ceasing to be a book. Four titles say so, on four shelves, in five forms each, and every one of them can be opened to the evidence under any sentence.

Acknowledgements

This work owes a debt to Professor Mark Liberman of the University of Pennsylvania, a veteran of Bell Labs, who years ago built a pioneering Emacs reader for studying Spanish that showed the dictionary translation of every Spanish word. That reader is the origin of the idea that the right tool for reading a foreign text is one in which every word can be asked, and it is what inspired me to package my Obsidian vault approach as Emacs again. Texinfo was my own first hypertext approach to documentation, in Philadelphia in 1994; it is a pleasure to find, thirty-two years later, that an Info manual is still a fine way to read a poem.

The texts of the four titles are the work of their authors, translators, and the volunteers of Project Gutenberg and Lib.ru; the dictionaries, of the Wiktionary and FreeDict communities and of William Whitaker; the code, of the Sail project. The readers, builders, and validators are at github.com/firstpair/firstpair, and the Dante edition at github.com/firstpair/dante.