Greenhouse Sentinel

Product Handbook and Docs-as-Code Portfolio Demonstration

Fictional product · Real documentation workflow · Katie Kearns · 2026

This public sample contains no customer, proprietary, export-controlled, or classified information.

Greenhouse Sentinel Documentation Demo

A fictional product with real documentation engineering behind it.

This is a public, self-documenting portfolio sample by Katie Kearns. Greenhouse Sentinel is a small fictional controller that watches three environmental signals and recommends a response before plants are stressed. Its limited scope makes the product documentation quick to understand while leaving enough structure to demonstrate a production-minded docs-as-code workflow.

Start the Product Tour

See How the Repo Works

Why This Demo Exists

This site is both a technical-writing sample and a reusable reference implementation. It shows how a documentation team can keep content reviewable in Git, publish a searchable site, customize presentation without rewriting source, detect broken navigation and cross-references, and generate portable Word and PDF deliverables.

Everything needed to evaluate the sample is documented on this site. The public GitHub repository provides the source and commit history for reviewers who want to inspect the implementation.

One Source, Several Useful Outputs

Authors maintain one set of accessible Markdown files. Automated checks verify navigation, internal links, anchors, and basic authoring rules. The publishing workflow then generates three outputs from that same reviewed source:

1. this searchable Zensical site
2. a Microsoft Word handbook
3. a PDF handbook

The Word and PDF files are not separately maintained copies. When an approved Markdown change is merged, automation rebuilds all three outputs together.

Download the Word handbook

Download the PDF handbook

What This Repository Demonstrates

Each feature is implemented in the public repository, not merely described in the portfolio narrative.

Feature	Made With	Why It Helps
Single-Source Authoring	One reviewed set of Markdown files under `docs/`	Authors change content once instead of reconciling separate website, Word, and PDF copies
Explicit Navigation	An ordered `nav` list in	Readers get a deliberate

Feature	Made With	Why It Helps
	`zensical.toml` plus a check for unlisted pages	sequence, and reviewers can trace every published page to its configuration
Stable Cross-References	Deliberate anchors on important targets, as shown by Decision States, plus automated fragment checks	Heading edits cannot silently strand important links
Automated Validation	`scripts/validate_docs.py`, which checks navigation, files, links, anchors, heading structure, and image text	Routine defects are found in seconds and resolved before publication
Continuous Integration and Publishing	A GitHub Actions workflow that repeats validation and generation from a clean copy of each proposed change	Every release follows the same recorded process, improving consistency, accuracy, and traceability
Accessible Authoring	Source standards, validation rules, accessible theme overrides, and a release checklist	Accessibility is addressed while content is written and reviewed instead of as a late repair
Replaceable Branding	CSS variables, a separate logo asset, and publishing configuration outside the Markdown	An organization can change the visual identity without rewriting or forking technical content
Multi-Format Delivery	Zensical for HTML, a Python export script for Word, and headless LibreOffice for PDF	Teams serve web and portable-document readers without maintaining three competing sources
Self-Documentation	Implementation notes, a complete build chapter, contributor guidance, and links to the actual configuration and scripts	Another writer can inspect, explain, reproduce, or adapt the workflow without undocumented setup knowledge

See How This Repository Works for a closer look at the implementation and benefit of each feature.

Architecture at a Glance

```
One reviewed Markdown source
          |
          +--> validation --> pull-request and release gates
          |
          +--> Zensical --> searchable HTML site
          |
```

```
            +--> export script --> Word handbook --> PDF handbook

   Brand tokens change presentation without changing source meaning.
```

The renderer can change while the release requirements remain stable. Explicit navigation, valid links and anchors, accessible structure, reproducible builds, and reviewable changes remain required.

The Fictional Product in 30 Seconds

Greenhouse Sentinel samples temperature, humidity, and soil moisture every 60 seconds. A rule engine assigns the greenhouse a state: **normal**, **watch**, or **act**. The operator dashboard explains the reading and links each alert to a documented response.

> **Portfolio disclosure.** Greenhouse Sentinel does not exist. Its behavior, data, and interface are intentionally simple and invented. The repository exists to demonstrate documentation architecture, authoring, validation, CI, publishing, and multi-format delivery without confidential information.

What Is Included

- **Product documentation** proves the workflow can support task-based technical content. Start with the quick start, operating guide, and architecture.
- **Repository documentation** reveals how the example is assembled and governed. Read How This Repository Works and Build It from Scratch.
- **Contributor guidance** makes quality expectations explicit and repeatable. Review the writing guide, accessibility standard, and release checklist.
- **Generated deliverables** prove that one reviewed source can support several channels. Use the Word and PDF download buttons above to inspect the portable versions.

Public Portfolio Scope

All product names, specifications, incidents, commands, and workflows in this demonstration are invented. The repository contains no Peraton-specific, customer, proprietary, export-controlled, or classified information. “ThreatBoard-style” describes the general docs-as-code pattern demonstrated by this sample, not copied content or branding.

Katie Kearns created this sample as a technical-writing and documentation-systems portfolio project. The code and documentation are available under the MIT License.

Quick Start

This walkthrough connects a fictional Sentinel hub and confirms that its three sensors are reporting. No hardware is required. The commands are illustrative.

Before You Begin

You need a Sentinel hub, one temperature and humidity probe, one soil probe, and access to the greenhouse Wi-Fi network.

Connect and Verify

1. Place the hub above the irrigation line and away from direct spray.
2. Connect both probes before applying power.
3. Join the temporary network named `sentinel-setup`.
4. Open `http://sentinel.local` and choose the greenhouse network.
5. Select **Run sensor check**.

A successful check reports all three readings and changes the hub indicator from amber to green.

```
temperature   24.1 C   ready
humidity      58 %     ready
soil          41 %     ready
```

Understand the First Result

The initial state should be **normal**. The controller waits for three consecutive samples before changing state, which prevents a single noisy reading from creating an alert. See Decision states for the complete rule.

If a sensor remains unavailable, follow A sensor reports no data.

> **Feature in action: task-oriented linking.** The link text names the reader's problem instead of saying “click here.” The validator rejects common context-free labels and verifies the stable troubleshooting anchor. This makes the reference clearer in the page, a screen reader's links list, Word, and PDF.

Next Step

Continue to the daily operating routine.

Operating Guide

The operating routine is designed for a volunteer who checks the greenhouse at opening and closing.

Daily Check

1. Confirm the dashboard timestamp is less than two minutes old.
2. Review the current state and its explanation.
3. Compare soil moisture across the three most recent samples.
4. Acknowledge any completed response.
5. Record an observation only when local conditions differ from the sensor data.

Respond to a State

Normal

No intervention is required. Continue the scheduled check.

Watch

Inspect the greenhouse within 30 minutes. Confirm the sensor is unobstructed and compare the reading with a handheld instrument when one is available.

Act

Follow the response displayed with the alert. Ventilate for high temperature, inspect irrigation for low soil moisture, or check airflow for sustained high humidity. Never bypass a physical safety control.

Acknowledge an Alert

Acknowledgement records that a person saw the alert. It does not change the sensor reading or decision state. Add a short note that states what you observed and what you did.

Good: `Opened roof vent; temperature fell to 27 C within 12 minutes.`

Avoid: `Fixed.`

For the system behavior behind these states, see Decision states.

> **Feature in action: stable cross-reference.** This source link points to `architecture.md#decision-states`. The file path keeps the link useful in GitHub and other Markdown tools. Zensical rewrites it for the published site. The destination has an explicit `decision-states` anchor, so minor heading edits do not break the reference. The independent validator confirms that both the file and anchor exist before publication.

Architecture

Greenhouse Sentinel separates measurement, evaluation, presentation, and response guidance so each part can be tested and explained independently.

System Boundary

The hub receives local sensor readings and produces a recommendation. It does not operate pumps, vents, heaters, locks, or other physical equipment. A person remains responsible for any physical action.

```
Sensors -> sample normalizer -> rule engine -> local event store -> dashboard
                                      |
                                      +-> response guidance
```

Components

- **Sample normalizer:** converts each supported sensor value to a consistent unit and marks stale data.
- **Rule engine:** evaluates the most recent valid samples against versioned thresholds.
- **Event store:** retains 30 days of readings, state changes, and acknowledgements on the hub.
- **Dashboard:** presents current conditions, recent trends, and the reason for the current state.

Decision States

The engine assigns one of three states after three consecutive samples meet the same condition.

State	Example condition	Operator expectation
Normal	All readings are inside target ranges	Continue scheduled checks
Watch	One reading is near a threshold	Inspect within 30 minutes
Act	One reading exceeds a threshold	Follow the displayed response

> **Feature in action: portable table.** This is a semantic Markdown table because the content has repeated, comparable fields. Zensical renders it responsively. The export script maps it to a real Word table with a header row. Ordinary explanatory prose remains outside tables for readability and accessibility.

The three-sample rule reduces noise without hiding sustained change. Thresholds in this demo are illustrative, not horticultural advice.

Data Retention

The hub stores 30 days of samples and event history. An export contains timestamps, normalized readings, state transitions, acknowledgements, and the rule-set version. It does not contain names unless an operator voluntarily enters one in a note.

Failure Behavior

If one sensor becomes stale, the dashboard reports **sensor unavailable** and does not infer a replacement value. The last valid measurement remains visible with its timestamp. See Troubleshooting.

Troubleshooting

Start with the visible symptom. Preserve timestamps and exact messages when escalating a problem.

A Sensor Reports No Data

1. Confirm the cable is fully seated and dry.
2. Check whether the dashboard timestamp continues to advance for other sensors.
3. Disconnect power, reconnect the sensor, and restore power.
4. Wait three sampling intervals.

If the sensor is still unavailable, record its port, the last valid timestamp, and the hub status indicator.

The Dashboard Is Unavailable

Confirm that your device and the hub are on the same network. Then open `http://sentinel.local` in a new browser window. If the page still does not load, restart the hub once and wait two minutes.

A State Seems Incorrect

Review the three most recent samples and compare them with the decision-state rule. A state can lag a single new reading because Sentinel requires three consecutive samples.

> **Warning.** Troubleshooting the fictional dashboard never overrides physical greenhouse safety procedures.

How This Repository Works

This page is the guided tour of the artifact you are reading. Every feature points to a real, inspectable implementation in the repository.

For the complete construction sequence, continue with Build It from Scratch.

Author Once in Markdown

All maintained content lives under `docs/`. Product guides and repository guidance use the same lightweight syntax, review process, and validation rules. Each page has one level-one heading. Sections follow a logical hierarchy. Relative links keep the content portable between source review and rendered output.

Follow One Authoring Process

An author works with the Markdown source rather than editing the generated site, Word file, or PDF.

1. **Start with a reader need.** Identify the task, question, or decision the page must support.
2. **Find the maintained source.** Edit the existing Markdown page or create a focused new page under `docs/`.
3. **Write and connect the content.** Use headings, descriptive relative links, stable anchors when needed, accessible tables, and verified examples.
4. **Run the local checks.** The validator checks navigation, files, anchors, headings, image text, and link wording. Zensical builds a local preview from the same source.
5. **Review the rendered result.** The author checks the page as a reader would, including navigation, responsive layout, keyboard behavior, and any generated Word or PDF pages affected by the change.
6. **Open a pull request.** Reviewers see the exact source diff, the automated results, and the generated artifacts. They do not edit a separate Word copy.
7. **Merge the approved change.** GitHub Actions repeats the checks, rebuilds every output, and publishes the live site from the accepted source.

The Markdown commit is the maintained record. The website, Word handbook, and PDF are outputs of that record.

Divide Human and Automated Work

Human judgment	Automated work
Decide what readers need and what belongs in scope	Confirm every configured navigation file exists
Verify technical meaning and release accuracy	Detect missing pages, files, links, and anchors
Write procedures, explanations, warnings, and troubleshooting guidance	Enforce basic heading, image-text, and link-label rules
Decide whether a table, note, example, or cross-reference helps	Build the Zensical HTML site from Markdown
Review accessibility, usability, and visual quality	Generate the Word handbook and convert it to PDF

Human judgment	Automated work
Approve the change	Check that expected artifacts exist and deploy the site

Automation handles repeatable questions with objective answers. Authors and reviewers remain responsible for accuracy, usefulness, safety, and clarity.

Control the Information Architecture

`zensical.toml` contains an explicit `nav` list. This makes the intended order and grouping reviewable instead of relying on a renderer to infer structure from filenames. The validation script also fails when a Markdown page is missing from navigation.

Keep Cross-References Stable

High-value targets use deliberate HTML anchors such as `decision-states` in the architecture guide. `scripts/validate_docs.py` resolves every internal Markdown link and fragment before publication. Changing a heading can no longer silently strand an important link.

The product pages include labeled **Feature in action** notes beside representative examples. These notes expose the implementation without interrupting the primary task for readers who only need the fictional product instructions.

See Each Authoring Feature Work

The repeated fields make these examples a semantic table: each row identifies a feature, its implementation, and its practical value.

Feature	Made With	Why It Helps
Cross-Reference	A deliberate `decision-states` anchor plus validation of the source file and fragment	Important links survive nearby heading edits, and broken targets stop the release
Descriptive Link	Task-specific link text plus a validator that rejects common context-free labels	Readers and screen-reader users can understand the destination out of context
Semantic Table	Markdown table syntax, accessible HTML, and real Word tables with repeating header rows	Relationships remain clear in HTML, Word, and PDF without separate copies
Admonition	A labeled `!!! info` block plus styling in the publishing layer	Important context stands out while the source stays readable and reviewable

Feature	Made With	Why It Helps
Code Block	Fenced Markdown, site copy support, and an explicit monospaced export style	Commands retain spacing and can be copied with fewer transcription errors
Explicit Navigation	The ordered `nav` list in `zensical.toml` plus checks for missing and unlisted pages	Page order is intentional, reviewable, and traceable instead of inferred from filenames
Brand Tokens	Purpose-based CSS variables and a separate logo asset outside the Markdown	Teams can rebrand consistently without changing technical meaning
Automated Validation	A renderer-independent Python script that checks navigation, structure, images, files, links, and anchors	Fast, repeatable checks free reviewers for judgment-heavy work and leave pass-or-fail evidence
Continuous Integration	`.github/workflows/docs.yml`, which repeats validation and publishing for proposed and accepted changes	The workflow catches environment failures and records exactly which version passed
Multi-Format Publishing	One navigation order, Zensical for HTML, a Python Word export, and LibreOffice PDF conversion	The website, Word handbook, and PDF stay synchronized from one maintained source

Separate Content from Presentation

`docs/assets/stylesheets/brand-2026.css` defines brand tokens and a small set of presentation rules. `docs/assets/images/mark.svg` contains the fictional mark. A new organization can replace the visual layer without editing product meaning. Try the branding exercise.

Validate Before Rendering

`scripts/validate_docs.py` checks:

- every configured navigation target exists
- every maintained Markdown page appears in navigation
- internal files and anchors resolve
- each page has exactly one level-one heading
- images have useful alternative text
- common inaccessible link labels are rejected.

`zensical build --strict` requests the renderer's strict behavior. Zensical 0.0.23 currently reports that strict mode is unsupported, so the custom validator is the enforceable gate today. It is deliberately independent of the renderer so core quality checks remain if the publishing engine changes.

Build and Review Automatically

`.github/workflows/docs.yml` runs validation, builds the Zensical site in strict mode, generates Word output, converts it to PDF, performs basic artifact checks, and uploads the results for review. Pull requests get the same gates as the default branch.

Publish Several Channels

Zensical creates the searchable HTML experience. `scripts/export_handbook.py` assembles the same maintained Markdown into a styled Word handbook with a **Page X of Y** footer. CI converts that Word file to PDF and preserves the footer numbers. These are publishing transformations, not separate authoring silos.

Treat Accessibility as an Authoring Constraint

The accessibility standard covers heading order, descriptive links, alternative text, tables, keyboard focus, zoom, contrast, and reduced motion. CSS enhances focus visibility and avoids essential animation.

Release with Evidence

The release checklist joins automated checks with human review: wording, task completeness, responsive behavior, keyboard use, zoom, and generated-file inspection.

Build It from Scratch

This chapter records every step used to construct this repository. Follow it to reproduce the demo or adapt the workflow for another small documentation set.

1. Define the Public Boundary

Choose a fictional product with enough behavior for procedures, concepts, reference material, troubleshooting, and cross-references. Record the release rule before drafting: no employer, customer, proprietary, export-controlled, or classified information. Greenhouse Sentinel is intentionally fictional and recommends—but never performs—physical actions.

2. Create an Independent Repository

Use a dedicated repository so its dependencies and automation run only when this project changes.

```
.
├── .github/workflows/docs.yml
├── deliverables/
├── docs/
│   ├── about/
│   ├── assets/
│   ├── contributing/
│   └── product/
├── scripts/
├── README.md
├── requirements.txt
└── zensical.toml
```

Ignore `site/`, virtual environments, caches, and local QA files. Keep portfolio-ready files in `deliverables/`.

3. Design the Information Architecture

Create three reader journeys: **Product guide** for realistic technical content, **How this repository works** for implementation details, and **Contribute** for authoring and quality standards. Write their order and labels explicitly in the `nav` array in `zensical.toml`.

4. Configure the Renderer

Pin Zensical in `requirements.txt`. Set the site identity, public and repository URLs, navigation, theme features, light and dark palettes, Markdown extensions, and custom stylesheet in `zensical.toml`. Use links to Markdown files so source previews and renderers can both resolve them.

5. Write a Thin but Complete Product Set

Create the product pages in reader order:

1. `quickstart.md` establishes the first successful outcome.
2. `operations.md` documents a repeatable task.
3. `architecture.md` explains boundaries, components, rules, and retention.

4. `troubleshooting.md` begins with observable symptoms and preserves escalation evidence.

Use one level-one heading per page, sequential subheadings, direct instructions, expected results, and honest fictional-product limitations.

6. Add Stable Cross-References

Add deliberate lowercase anchors before high-value targets:

```
<a id="decision-states"></a>
## Decision States
```

Link with a relative source path and stable fragment:

```
[Decision states](architecture.md#decision-states)
```

This decouples important links from incidental heading punctuation.

Explain the feature where readers encounter it. This demo uses a labeled `!!! info` note after the representative cross-reference. The note identifies the source syntax, published behavior, accessibility benefit, and validation rule. Apply the same pattern selectively to tables, code blocks, admonitions, navigation, branding, and exports so the repository demonstrates rather than merely claims each capability.

7. Build Independent Validation

Create `scripts/validate_docs.py` with Python standard-library dependencies. Read `zensical.toml`, flatten navigation, discover maintained Markdown, extract headings and anchors, and resolve every local link relative to its source.

Fail when navigation names a missing file, a maintained page is omitted, a file or anchor does not resolve, a page lacks exactly one level-one heading, heading levels skip, an informative image has no alternative text, or a link uses context-free wording such as “click here.” Keep this separate from Zensical so renderer changes do not weaken the release gate.

8. Add Replaceable Branding

Create `docs/assets/stylesheets/brand-2026.css` with purpose-based tokens for primary, accent, surface, text, focus, and radius values. Add a compact accessible SVG mark. Use the tokens for visible focus, mobile button stacking, readable line lengths, and reduced-motion behavior. Do not place brand decisions in Markdown.

9. Generate Word from Reviewed Source

Create `scripts/export_handbook.py`. Read the same navigation list as the site and process pages in that order. Map Markdown headings to real Word heading styles, sequences and bullets to list styles, code to a monospaced style, and Markdown tables to Word tables. Apply explicit page geometry, typography, spacing, colors, public document properties, and dynamic `PAGE` and `NUMPAGES` footer fields so every page displays **Page X of Y**.

10. Generate PDF

In CI, convert the Word handbook with headless LibreOffice. Check that Word and PDF files exist and are non-empty. Render every page to images for visual review. Inspect the pages for clipping, overlap, broken tables, awkward breaks, missing glyphs, misplaced page furniture, and correct footer page numbers.

11. Automate the Workflow

Create `.github/workflows/docs.yml` for pushes, pull requests, and manual runs. The workflow automates the mechanical publishing sequence:

1. Check out the exact Git commit under review.
2. Install the dependency versions pinned by the repository.
3. Run `scripts/validate_docs.py` against the Markdown and navigation configuration.
4. Build the complete Zensical site from the reviewed source.
5. Run `scripts/export_handbook.py` to assemble the Markdown pages in explicit navigation order.
6. Create the Word handbook with real heading, list, table, and document styles.
7. Convert the Word handbook to PDF with headless LibreOffice.
8. Confirm that the site, Word file, and PDF exist and are non-empty.
9. Upload the site and portable files as reviewable build artifacts.
10. On the `main` branch, upload the site to GitHub Pages and deploy the live URL.

The workflow does not decide whether an explanation is correct or whether a procedure is useful. Those decisions remain with the author and reviewers. Automation answers repeatable questions such as whether a target exists, whether the build succeeds, and whether all expected outputs were produced.

Zensical 0.0.23 currently warns that strict mode is unsupported. Retain the flag for forward compatibility, but make the independent validator the enforceable gate.

12. Test the Site

Build and serve `site/`. Test at 320, 360, 390, 768, 1024, and 1440 CSS pixels. At every width, verify `scrollWidth <= clientWidth`. Test the skip link, navigation, search, theme control, links, buttons, visible focus, heading order, descriptive links, 200% zoom, long-label wrapping, touch targets, light and dark contrast, and reduced motion.

13. Review for Public Release

Complete the release checklist. Inspect source history, rendered pages, generated files, document properties, repository description, and README for sensitive or employer-specific material. Confirm every limitation is stated plainly.

14. Publish the GitHub Repository

Initialize Git in this folder, set `main` as the default branch, commit the verified files, create a public repository named `greenhouse-sentinel-docs`, and push only this folder. Require the documentation workflow before merge when account settings allow it. Optionally deploy `site/` with GitHub Pages.

15. Link from the Portfolio

After publication, add a small portfolio entry that links to this repository and optionally its live site. Describe the reader problem and demonstrated capabilities. Do not embed this repository's build into the portfolio build.

16. Update and Release

The normal authoring cycle is smaller than the initial repository setup.

Author the Change

1. Start with a reader problem or a verified product change.
2. Edit the relevant Markdown under `docs/`. Do not edit `site/`, Word, or PDF because those files are generated outputs.
3. Add a new page to `zensical.toml` when the change introduces a new topic.
4. Add or update relative links and stable anchors when other pages depend on the content.
5. Keep branding changes in the stylesheet, logo asset, or publishing configuration.

Check the Change Locally

1. Run `python scripts/validate_docs.py`.
2. Build or serve the Zensical site.
3. Read the rendered page in context, not only the source diff.
4. Regenerate Word and PDF when the change affects portable output.
5. Complete editorial, accessibility, responsive, and generated-file review in proportion to the risk of the change.

Review and Publish

1. Commit only the intended source and configuration changes.
2. Open a focused pull request that explains the reader need and the checks performed.
3. Let CI repeat validation and generate review artifacts from a clean environment.
4. Have the appropriate people review technical meaning, writing quality, accessibility, and presentation.
5. Merge only after the automated gates and human review pass.
6. Let the `main`-branch workflow rebuild every output and deploy GitHub Pages.

This creates a simple ownership boundary. People author and approve meaning. The repository automates validation, transformation, packaging, and deployment.

Branding

Branding belongs in the publishing layer. The source should still make sense as plain Markdown, in GitHub preview, in Word, and in a different web theme.

Five-Minute Rebrand

1. Open `docs/assets/stylesheets/brand-2026.css`.
2. Replace the values under `:root` for the primary, accent, surface, text, and focus colors.
3. Replace `docs/assets/images/mark.svg` with a square logo that has a meaningful filename.
4. Update `site_name`, `site_author`, and `copyright` in `zensical.toml`.
5. Run the validator and a strict production build.
6. Test the rendered site at the widths in the release checklist.

Token Approach

The stylesheet maps brand decisions to purpose-based tokens such as `--brand-primary` and `--brand-focus`. Components use those tokens rather than repeating literal colors. This keeps a rebrand small, reviewable, and consistent.

Content That Should Not Be Branded

Do not embed organization names in reusable technical explanations, hard-code colors into diagrams without a legend, or use a logo as the only way to identify a page. Brand changes should not alter instructions, safety meaning, heading order, or link destinations.

Publishing

The repository treats publishing as a reproducible transformation of reviewed source.

HTML with Zensical

Zensical is the preferred renderer because it supports explicit navigation, a searchable static site, and a modern responsive theme. The checked-in configuration keeps the local preview and CI build aligned.

```
zensical build --strict
```

Version 0.0.23 currently warns that strict mode is unsupported. The command is retained for forward compatibility, but the independent documentation validator—not that flag—is the required link, anchor, structure, and navigation gate.

Word and PDF

The export script reads the explicit navigation list so portable outputs follow the same order as the site. It applies a restrained document style, real heading levels, descriptive link text, and a running footer with dynamic **Page X of Y** fields. LibreOffice preserves those page numbers when it creates the PDF.

```
python scripts/export_handbook.py
```

CI uses LibreOffice to convert the generated Word file to PDF. Both files are checked for existence and plausible size before publication.

The website, Word handbook, and PDF handbook are three presentations of the same reviewed Markdown source. Authors do not update any of them separately. After an approved Markdown change is merged, the workflow rebuilds the site, generates Word, converts Word to PDF, verifies all three outputs, and publishes them together.

Download the Word handbook

Download the PDF handbook

Renderer-Independent Gates

A replacement renderer is acceptable only if the release continues to verify navigation completeness, target files, anchors, accessible structure, responsive layout, and portable outputs. Tool choice can evolve. The acceptance criteria stay explicit.

Writing Guide

Write for the person completing a task, then make the implementation easy to review.

Structure

- Use one level-one heading per page.
- Keep heading levels sequential. Do not skip from level two to level four.
- Begin procedures with the reader's goal and prerequisites.
- Use numbered lists for sequences and bullets for unordered choices.
- Put reusable, high-value targets behind deliberate lowercase anchors.

Language

- Prefer direct verbs: **Select Save**, not **The Save button should be selected**.
- State expected results after important steps.
- Use consistent terms for components and states.
- Avoid “click here,” “read more,” and other links that lose meaning out of context.
- Label fictional values and limitations honestly.

Cross-References

Use relative Markdown paths and include the stable fragment when linking to a section:

```
[Decision states](../product/architecture.md#decision-states)
```

Run `python scripts/validate_docs.py` before opening a pull request.

Authoring Workflow

1. Identify the reader goal and confirm the product information.
2. Edit Markdown under `docs/`. Treat the website, Word handbook, and PDF as generated outputs.
3. Update explicit navigation when adding or moving a page.
4. Add descriptive links, stable anchors, images, and examples needed by the topic.
5. Run the independent validator and build a local Zensical preview.
6. Review the rendered page for content, accessibility, navigation, and responsive behavior.
7. Open a focused pull request and describe the checks performed.
8. Address human review. Merge only after the required automated gates pass.

After merge, GitHub Actions rebuilds the site and portable files. It then deploys the approved site to GitHub Pages.

Accessibility

Accessibility is part of authoring and release, not a cleanup pass after publication.

Authoring Standard

- Use a logical heading hierarchy and one clear page title.
- Write descriptive link text that makes sense in a list of links.
- Add concise alternative text to informative images. Hide decorative imagery from assistive technology in templates.
- Give tables a header row and use tables only for genuinely tabular relationships.
- Do not use color as the only way to communicate state.
- Keep instructions independent of visual position such as “the green button on the right.”

Rendered-Site Checks

- Navigate every interactive control with a keyboard and confirm focus is visible.
- Verify the skip link reaches the main content.
- Confirm text and controls remain usable at 200% browser zoom.
- Test at 320, 360, 390, 768, 1024, and 1440 CSS pixels.
- At every width, confirm the page has no horizontal overflow.
- Respect `prefers-reduced-motion`. Do not require animation to understand content.

Portable-Output Checks

Inspect every generated Word and PDF page for clipped text, broken tables, poor page breaks, missing glyphs, and unclear link text. Confirm Word uses real heading styles so readers can navigate by structure.

Release Checklist

Automated Gates

- [ ] The independent documentation validator passes.
- [ ] `zensical build --strict` succeeds.
- [ ] Every maintained page is present in explicit navigation.
- [ ] Internal files and anchors resolve.
- [ ] Word and PDF outputs are generated and pass artifact checks.

Editorial Review

- [ ] The change answers a reader goal and states expected results.
- [ ] Terms, labels, units, and cross-references are consistent.
- [ ] No confidential, customer, proprietary, export-controlled, or classified information is present.
- [ ] Fictional values and limitations are identified.

Accessibility and Responsive Review

- [ ] Heading order and landmark structure are logical.
- [ ] Keyboard focus is visible and the skip link works.
- [ ] Text remains usable at 200% zoom.
- [ ] Pages have no horizontal overflow at 320, 360, 390, 768, 1024, and 1440 CSS pixels.
- [ ] Long labels, buttons, navigation, email addresses, and graphics wrap or crop safely.
- [ ] Motion is reduced when the operating-system preference requests it.

Generated-File Review

- [ ] Every Word and PDF page has been visually inspected.
- [ ] Headings, lists, tables, links, headers, and footers render correctly.
- [ ] File names, document properties, and visible content are suitable for public release.