A RESOURCE FOR DESIGN SYSTEM AND DOCS TEAMS

Promptable docs template.

One page per component, written for readers that cannot ask back.

Rules first. Everything enumerated. Every example compiled.

25	17	01
writing rules, each checkable	sections in a fixed order	page per component

Docs are now read by two audiences at once.

Agents reach your documentation through MCP servers, Storybook manifests, llms.txt files and retrieval indexes. Each of them ends up serving the same thing: a component page, often as a single chunk cut from the middle. A page written for a person who can scroll, infer and ask a colleague fails that reader. The fix is not a second set of docs for machines; it is one page that works for both.

The evidence on formats is still thin and points in different directions. Vercel found that a small always-loaded docs index in AGENTS.md reached 100% on its Next.js evals, where an on-demand skill reached 79% even with explicit instructions [S07]. Atlassian found its MCP server beat a single DESIGN.md file on tokens used and design coverage [S08]. Both are vendor evals of single tasks. What they agree on is that the component page itself has to be excellent retrieval material, with a small index pointing to it.

This guide gives that page a fixed structure and 25 writing rules, and the kit makes them checkable: a template, a frontmatter schema, a linter, a generator that writes the API table from the Field Guide 02 contract, a script that extracts every example for the compiler, an llms.txt and ready-made agent instruction files.

Version 1.0 / Sources checked 24 September 2026

Field Guide 04 of the Design × AI series. The worked example is the same Button as Field Guides 02 (contract) and 03 (tokens); its examples typecheck against the contract's types.

Practical guidance, not a standard: none exists for component doc structure or frontmatter. Prepared with AI assistance and edited by hand.

START HERE

Pick your route, then read the labels.

Apply the template to one component end to end before rolling it out. The rules are ordered by where they bite: structure, rules, API, examples, accessibility, delivery.

Writing a new page

Copy `component-doc.template.md`, fill it top to bottom and run the linter. Generate the API table from the contract rather than typing it.

Converting existing docs

Sections 01 and 02 first: fixed headings and a Rules section do most of the work. Move version history into Old patterns.

Opening docs to agents

Section 06: publish a `.md` twin, list it in llms.txt, and paste the AGENTS.md snippet into consuming repos.

Proving it works

W25: three agent tasks per component, run without docs and with docs. Keep the ones that fail as regression tests.

Label	Meaning
`LINT`	Checked by `lint-component-doc.mjs` in the kit.
`CI`	Checked by another kit script or CI job: the API generator, the code extractor, a staleness job.
`AGENT`	Exists so an agent retrieves, reads or follows the page correctly.
`PRACTICE`	A working method with a review signal rather than a hard gate.

> **The page is the interface.**
>
> MCP tools, Storybook manifests, llms.txt links and retrieval indexes all serve component pages. Improve the page and every channel improves with it.

SECTION 01

Structure the page for chunked reading.

Retrieval often returns one section, not the page. Every section has to stand on its own and sit where a reader, human or model, expects it.

Suggested owners: Docs owner + design-system lead

- [ ] **Frontmatter carries the machine metadata**

  W01 LINT

  Identity, package and import, versions the page applies to, lifecycle status, owner, source URLs for the HTML page and its `.md` twin, and a last-verified stamp. Flat enough to index as metadata filters.

  Evidence: `doc-frontmatter.schema.json`, validated by the linter.

  Field names from Shopify (`source_url`), Mintlify, Custom Elements Manifest and Storybook. S10, S12.

- [ ] **The H1 and a one-sentence summary open the page**

  W02 LINT

  A blockquote summary of 30 words or fewer that names the component and its one job. It is identical to the frontmatter `summary`, and it is the chunk head every retriever sees first.

  Evidence: Linter: H1 equals `title`; blockquote equals `summary`.

  Mirrors the llms.txt H1 and blockquote pattern. S01.

- [ ] **Rules come first**

  W03 LINT

  `## Rules` is the first H2. Constraints before prose, because the Rules chunk has to stand alone when it is the only one retrieved.

  Evidence: Linter: fixed H2 order.

  Recommended structure.

- [ ] **Fixed heading text, fixed order**

  W04 LINT

  Seventeen H2 sections with the template's exact wording, on every component page. Predictable headings give predictable anchors and chunks.

  Evidence: Linter: H2 list compared with the template.

  Mintlify: skipped or renamed headings hide how sections relate. S11.

- [ ] **Every section names the component**

  W05 LINT AGENT

  The first sentence of each section says "Button", never "it" or "this component". A chunk read on its own has no antecedent.

  Evidence: Linter warning when a section's first sentence omits the title.

  Mintlify on vague pronouns. S11.

One term per concept

W06 PRACTICE

Pick `tone` and never also say variant, kind or appearance for the same thing; pick prop, never option or param. Keep the list in a glossary and lint for banned synonyms.

Evidence: A glossary with banned synonyms; the Does not exist list.

Anthropic: use consistent terminology. S05.

Nothing hides in tabs, images or widgets

W07 PRACTICE

The `.md` twin contains every variant and every example as plain Markdown, and no rule exists only in an image or video.

Evidence: Diff the code blocks in the HTML page against the `.md` twin.

Mintlify structure guide. S11.

Concise by default

W08 LINT

Skip what the model already knows about the web or the framework. Keep a page under 500 lines; split larger ones.

Evidence: Linter warning over 500 lines.

Anthropic skill guidance, applied by analogy to docs. S05.

Write every section as if it were the only one retrieved, because often it is.

SECTION 02

Write rules a model can follow and a script can check.

The Rules section is the only text meant to be copied into agent rule files. Keep it short, keep its words exact and give every prohibition a way out.

Suggested owners: Docs owner + accessibility lead

- [ ] **Declare RFC 2119 and 8174 once**

  W09 PRACTICE

  State once, on the docs index and in llms.txt, that MUST, MUST NOT, SHOULD, SHOULD NOT and MAY carry their BCP 14 meaning only in capitals.

  Evidence: The boilerplate is present in llms.txt (the kit's example has it).

  RFC 8174. S03.

- [ ] **Three to ten rules, each with a keyword**

  W10 LINT

  Most important first. Each bullet carries exactly one normative keyword in capitals.

  Evidence: Linter: 3 to 10 bullets, each with a keyword.

  Recommended practice.

- [ ] **Every prohibition gives the reason and the replacement**

  W11 LINT AGENT

  "Button MUST NOT navigate to another page, because screen readers announce it as a button; use `Link` instead." A bare "don't" leaves the model to guess the alternative.

  Evidence: Linter: every MUST NOT and SHOULD NOT contains "because" and "use".

  Anthropic: give the reason, and say what to do instead. S04.

- [ ] **Capitals only in Rules**

  W12 LINT

  Outside Rules (and the "You MUST provide" accessibility list) write plain facts. Lowercase "should" and "must" read as advice, so avoid them too.

  Evidence: Linter: RFC keywords outside Rules are errors; lowercase normative words are warnings.

  RFC 8174: lowercase words keep their ordinary meaning. S03.

- [ ] **No shouting**

  W13 PRACTICE

  No CRITICAL, IMPORTANT! or bold-capital banners. RFC keywords are a vocabulary, not emphasis.

  Evidence: Grep for emphasis banners in review.

  Anthropic notes that newer models can over-trigger on aggressive emphasis. S04. No published eval shows that RFC capitals improve compliance; the case for them is clarity and checkability.

A rule without a replacement is a trap, not a rule.

SECTION 03

Enumerate the API, and name what does not exist.

Agents invent what the page leaves open. Close every set, generate the table from source and list the guesses you have already seen.

Suggested owners: Design-system lead + engineering lead

- [ ] **Generate the API table from the contract**

  W14 CI

  Hand-written prop tables drift. The kit writes the table from the Field Guide 02 contract between two marker comments and fails CI when the page differs.

  Evidence: `generate-api-table.mjs page.md contract.json --check`.

  Storybook recommends docgen for accuracy. S13.

- [ ] **Close every set in words**

  W15 CI AGENT

  "Valid values: `primary`, `secondary`, `ghost`, `destructive`. No other values are valid." The generator writes this sentence for every enum.

  Evidence: Generated API table.

  Enumerated sets make invented values provable. S09.

- [ ] **Name the hallucinations**

  W16 LINT AGENT

  A "Does not exist" list under Do and don't: the props, values and components agents keep guessing (`variant`, `shadow`, `PrimaryButton`), with the real alternative. Feed it from agent-eval failures.

  Evidence: Linter: the section is required.

  Ant Design's export allow-list with named non-existent components. S09.

- [ ] **One default, then the exception**

  W17 PRACTICE

  "Use Button. For navigation, use Link." Three equal options invite a coin toss.

  Evidence: When to use names exactly one default per task.

  Anthropic: provide a default with an escape hatch. S05.

Anything the page does not close, an agent will open.

SECTION 04

Examples that compile, and mistakes that teach.

Models copy examples more reliably than they follow prose. That makes every example a specification, so every example has to be correct.

Suggested owners: Docs owner + engineering lead

- [ ] **One complete canonical example first**

  W18 LINT

  Imports included, no `...`, no placeholder props, accessible name present. It is the example an agent copies by default.

  Evidence: Section order in the linter; the extractor compiles it.

  Recommended practice.

- [ ] **Every code block compiles in CI**

  W19 CI

  Extract the blocks and typecheck them against the version in `last_verified`. The kit's Button examples compile against the contract's own types, and an invented `tone` in any of them fails the build.

  Evidence: `extract-doc-code.mjs` then `tsc --noEmit`; blocks after "Don't:" are skipped because they are wrong on purpose.

  Kit scripts.

- [ ] **Every fence has a language**

  W20 LINT

  Fenced blocks with a language tag, never inline code for multi-token snippets.

  Evidence: Linter: untagged fences are errors.

  Mintlify: inline code can split into separate tokens. S11.

- [ ] **Do and don't pairs are symmetric**

  W21 PRACTICE

  Each Don't block is followed by one sentence of reason and a Do block that fixes the same case.

  Evidence: Review: matching counts of Do and Don't blocks.

  Recommended practice.

An example that does not compile teaches the wrong API with full confidence.

SECTION 05

Split what the component gives from what you owe.

Most accessibility failures in generated UI are consumer failures: the component was fine, the accessible name was missing.

Suggested owners: Accessibility lead + docs owner

- [ ] **Provides, You MUST provide, Keyboard interaction**

  W22 LINT AGENT

  Three fixed subsections. Link the APG pattern, keep the keyboard table in APG's shape and state the accessible-name requirement explicitly. Never recommend ARIA that duplicates native semantics.

  Evidence: Linter: all three subsections are required.

  APG pattern pages; APG: no ARIA is better than bad ARIA. S14.

The component can promise a role. Only the consumer can supply a name.

SECTION 06

Keep it fresh, findable and tested.

A correct page nobody retrieves, or a stale one that everybody does, is worse than no page. Stamp it, publish it where agents look, and measure it.

Suggested owners: Docs owner + platform owner

- [ ] **Version-conditional text lives in Old patterns**

  W23 LINT

  No "before v4", "as of 3.2" or "recently" in the main sections. Deprecated usage goes in one table: old usage, removed in, replacement.

  Evidence: Linter: version-conditional phrases outside Old patterns are errors.

  Anthropic: avoid time-sensitive information; keep an old-patterns section. S05.

- [ ] **Stamp it, twin it, list it**

  W24 LINT CI

  A machine-checkable `last_verified` (date, package version, method) with a staleness job; a `.md` twin advertised with `rel="alternate" type="text/markdown"`; and a link from an llms.txt that follows v2.

  Evidence: Linter warns on stamps older than 180 days (a local policy); a HEAD request shows the Link header.

  llms.txt v2 (August 2026). S01, S02.

- [ ] **Evaluate the page with agents**

  W25 PRACTICE

  Keep at least three agent tasks per component ("build a destructive confirm dialog with Button") with pass criteria. Run them without the docs, then with them, and after every significant edit.

  Evidence: An eval file per component, run in CI or on release.

  Anthropic: build evaluations before writing extensive docs. S05. Atlassian and Vercel publish results from evals of this kind. S07, S08.

Measure the page with the readers it is written for.

APPENDIX A

The page, section by section.

The fixed order the linter enforces, with the job of each section. Section titles are exact.

Section	Job	Written by
Frontmatter, H1, summary	Identity and the chunk head every retriever sees	Author
Rules	The normative contract; the only text copied into agent rule files	Author
When to use / When not to use	Choose this component, or name the alternative	Author
Import	The exact import line	Author
Canonical example	The default code an agent copies	Author, compiled in CI
API	Every prop, type, value and default, plus combination rules	Generated from the contract
Variants	Purpose of each value, when and when not	Author
States	How each state is set, what the component does, what you provide	Author
Composition	Allowed parents and children, forbidden nesting	Author
Accessibility	Provides, You MUST provide, keyboard table	Author with accessibility lead
Content guidelines	Label rules as Do and Don't text pairs	Content designer
Do and don't	Code-level mistakes with fixes; Does not exist	Author, fed by evals
Tokens	Tokens the component uses (Field Guide 03)	Author
Related	Sibling components with a use-instead-when note	Author
Old patterns	Deprecated usage and replacements, the only versioned text	Author
Verification	Commands and tools that check output mechanically	Author
Machine-readable sources	Contract, manifest, story IDs, token files	Author

examples/button.md (frontmatter and Rules)

```
---
id: button
title: Button
summary: Button triggers one action in the current view, such as saving or submitting a form.
package: "@acme/ui"
import: "import { Button } from \"@acme/ui\";"
applies_to_versions: "⩾4.2 <5"
status: stable
last_verified: { date: 2026-09-24, against_version: "4.6.1", method: ci-doc-tests }
contract: ./button.contract.json
---

# Button

> Button triggers one action in the current view, such as saving or submitting a form.

## Rules

- Button MUST have an accessible name: visible text, or `aria-label` when `iconOnly` is true.
- Button MUST NOT navigate to another page, because screen readers announce it as a button
  and users expect an action; use `Link` instead.
```

Abbreviated for print: the file uses multi-line YAML and one line per rule. The full page is in the kit and passes the linter with no errors or warnings.

APPENDIX B

Point agents at the page.

Instruction files are always loaded, so they carry a short router, not the docs. The llms.txt follows the v2 proposal: one H1, a blockquote, plain notes, then H2 sections of link lists.

llms.txt (excerpt)

```
# Acme Design System

> Acme Design System is the React component library, design tokens and usage guidelines for
> Acme web products. Package: `@acme/ui` 4.x.

The key words MUST, MUST NOT, SHOULD, SHOULD NOT and MAY in linked pages are to be interpreted
as described in BCP 14 (RFC 2119, RFC 8174) when, and only when, they appear in all capitals.

- `@acme/ui` does not export `Box`, `Stack`, `Container`, `Heading`, `PrimaryButton` or
`IconButton`.

## Components

- [Button](https://design.acme.example/components/button.md): Triggers one in-page action
- [Link](https://design.acme.example/components/link.md): Navigates to another page

## Optional

- [Complete documentation](https://design.acme.example/docs/llms-full.txt): Every page
concatenated
```

llms-full.txt is a platform convention, not part of the spec, so it sits under Optional.

agents/AGENTS.snippet.md

```
## UI components (Acme Design System)

- Build UI with `@acme/ui` 4.x. Icons come from `@acme/icons`.
- Prefer retrieval over recall: read the component page before using a component.
  Index: https://design.acme.example/docs/llms.txt
- Use only exported components and only the props and values each page lists.
- Style with semantic tokens (`var(--ds-color-*)`). Never use `--ds-core-*` or `--ds-comp-*`.
- Before finishing, run `npm run lint`, `npx tsc --noEmit` and the Storybook tests.
```

The kit also has the same router as a path-scoped Claude Code rule, a Copilot instructions file and a Cursor rule.

File	Read by	Scope
`AGENTS.md`	Codex, Copilot coding agent, Cursor, Claude Code and others	Nearest file in the tree
`.claude/rules/*.md`	Claude Code	`paths:` globs in frontmatter
`.github/instructions/*.instructions.md`	GitHub Copilot	`applyTo:` globs
`.cursor/rules/*.mdc`	Cursor	`globs`, `alwaysApply`

Verify file names and scoping against the tool versions you run; these conventions still move.

APPENDIX C

Run the checks.

Three scripts turn the rules into a build step. Output below is from the kit's Button page.

terminal

```
$ node scripts/generate-api-table.mjs examples/button.md button.contract.json --check
examples/button.md: API section matches the contract.

$ node scripts/lint-component-doc.mjs examples/button.md
1 page(s), 0 error(s)

$ node scripts/extract-doc-code.mjs examples/button.md .doc-tests
Extracted 5 code block(s) into .doc-tests
$ npx tsc --noEmit -p .doc-tests
(no output: every example compiles against the Button contract types)
```

Linter check	Severity
Frontmatter against `doc-frontmatter.schema.json`	error
H1 equals title; blockquote equals summary	error
Seventeen H2 sections, exact text, fixed order	error
3 to 10 rules, each with an RFC keyword; MUST NOT with because and use	error
RFC keywords outside Rules and You MUST provide	error
Untagged code fences; missing API markers	error
Version-conditional wording outside Old patterns	error
Missing Does not exist list or accessibility subsections	error
Lowercase should or must outside Rules	warning
Section's first sentence does not name the component	warning
last_verified older than 180 days; page over 500 lines	warning

KEEP WITH THE PAGE

Leave a page review record.

A record for one component page. It shows which checks ran against which package version, and what the agent evals said.

Component / page id	**Package version verified against**
Linter run (link)	**API table check against contract (link)**
Extracted examples compiled (link)	**Agent eval tasks and results (link)**
Does not exist list updated from evals	**Accessibility review (name, date)**
Owner	**Next review date**

> **A green linter is not a good page.**
> The linter proves the page is complete and consistent. Only the agent evals show whether it helps.

SOURCES / MAINTENANCE

Keep the guide current.

Sources checked 24 September 2026. Vendor eval figures are single-task results published by the vendors; treat them as signals, not benchmarks.

S01 / llms.txt proposal (v2, August 2026)
File structure, link relations and path scoping.
https://llmstxt.org/index.md

S02 / llms.txt changes
What v2 changed, including the Optional section.
https://llmstxt.org/changes.md

S03 / RFC 8174
Ambiguity of uppercase and lowercase in RFC 2119 keywords.
https://www.rfc-editor.org/rfc/rfc8174.txt

S04 / Anthropic, prompting best practices
Give reasons, say what to do instead, avoid aggressive emphasis.
https://platform.claude.com/docs/en/build-with-claude/prompt-engineering/claude-prompting-best-practices

S05 / Anthropic, Agent Skills best practices
Concise content, consistent terms, defaults, evaluations first, old-patterns sections.
https://platform.claude.com/docs/en/agents-and-tools/agent-skills/best-practices

S06 / AGENTS.md
The shared agent instruction file convention.
https://agents.md/

S07 / Vercel, AGENTS.md outperforms skills in our agent evals
Always-loaded docs index at 100% against a skill at 79% (January 2026).
https://vercel.com/blog/agents-md-outperforms-skills-in-our-agent-evals

S08 / Atlassian, testing DESIGN.md in practice
MCP server against DESIGN.md on tokens and design coverage (June 2026).
https://www.atlassian.com/blog/how-we-build/atlassians-design-md-is-here-what-we-learned-testing-portable-design-context-in-practice

S09 / Kaelig Deloumeau-Prigent, State of AI in Design Systems
Survey of 21 systems (July 2026); secondary source for per-system tooling.
https://state-of-ai-in-design-systems.netlify.app

S10 / Shopify, Polaris with MCP (markdown twin)
Frontmatter with title, description and source_url for html and md.
https://shopify.dev/docs/api/polaris/using-mcp.md

S11 / Mintlify, structuring documentation for AI and human readers
Headings, pronouns, tabs and inline code in chunked retrieval.
https://www.mintlify.com/blog/structure-documentation-AI-human-readers

S12 / Custom Elements Manifest
Structured component metadata that pairs with prose docs.
https://github.com/webcomponents/custom-elements-manifest

S13 / Storybook, writing docs for AI
Docgen, JSDoc descriptions and one concept per story.
https://storybook.js.org/docs/ai/best-practices

S14 / W3C, ARIA Authoring Practices: button pattern
Keyboard interaction and roles, states and properties.
https://www.w3.org/WAI/ARIA/apg/patterns/button/

Maintenance: recheck the llms.txt proposal, the agent instruction file conventions and the Storybook AI pages every quarter; all three moved in 2026. Update the PDF, HTML, Markdown and JSON together.