A RESOURCE FOR DESIGN SYSTEM AND DOCS TEAMS

Component doc generation prompt.

A first draft from the contract, checked before a person reads it.

Scripts write the facts. The model writes the prose. The eval decides.

25	10	01
checks, inputs to published page	eval checks on every draft	prompt for every component

Generate the draft, not the truth.

Component documentation is a good first job for a model: the inputs already exist as structured files and a draft is cheap to throw away. It is also easy to get wrong in ways nobody notices, such as an invented prop in an example or a keyboard shortcut the component never had. Agents then copy those mistakes into product code.

This guide splits the job. Scripts compute everything with one right answer, the frontmatter and the API table, from the contract. The model writes the prose around those facts into the 17-section page of Field Guide 04. An eval then runs the Field Guide 04 linter and seven checks against the contract and the component index before a person reads the draft.

The kit ships the prompt with variables, a filler, the eval and a Button page generated from the filled prompt. The structure follows Anthropic's guidance on long documents, XML tags and examples [S01]. Nothing in the kit calls a model.

Version 1.0 / Sources checked 24 September 2026
Field Guide 19 of the Design × AI series. Uses the Button from Field Guide 02 and the page format, linter and API generator from Field Guide 04. Anthropic's prompting and structured output docs checked 24 September 2026.

Practical guidance, not a standard. The sample page was generated once from the filled prompt by Claude (Opus 5.5) during authoring; Appendix B shows the one hand edit. Prepared with AI assistance and edited by hand.

START HERE

Pick your route, then read the labels.

Run the kit on the Button first. You will see the filled prompt, a reply that fails one check, and the edited page that passes all of them.

Documenting a new component

Write the contract first (Field Guide 02). Then run `fill-prompt.mjs` on the component folder, send `request.json`, and run the eval on the reply.

Converting old docs

Pass the old page with `--existing`. The prompt treats it as the lowest-authority document, so stale guidance loses to the contract and turns into a gap.

Adapting the prompt

Edit `prompt/component-doc.prompt.md` and `prompt/examples.md`, keep the variables, and rerun the Button as a regression test before you use it on anything else.

Deciding whether to automate it

Field Guide 21 scores doc drafting as assist (draft): the model drafts, a named owner publishes. This guide is the gate for that decision.

Read the labels before the checks.

Label	Meaning
`PROMPT`	Built into the prompt template or the filler script.
`SCRIPT`	Checked by a kit script: the filler, the eval or the Field Guide 04 tools.
`REVIEW`	Checked by a person, because no script can.
`PRACTICE`	A working method with a review signal rather than a hard gate.

the pipeline

```
1. component folder      contract, types, source, stories
2. fill-prompt.mjs       system.md, prompt.md, request.json
3. your own client       reply with <page> and <gaps>
4. eval-generated-doc    lint, API, contract facts, links, residue, gaps
5. owner review          only when the eval reports 0 errors
```

SECTION 01

Feed it sources, not memories.

A model fills every silence with general knowledge about buttons. Give it ranked documents, and compute what needs no model.

Suggested owners: Design-system lead + docs owner

- [ ] **The contract is the first document and the top authority**

  P01 PROMPT

  The Field Guide 02 contract goes in first, and the prompt says it wins every disagreement. Props, values, constraints, keyboard rows and anti-patterns come from it.

  Evidence: Document 1 in the filled prompt; instruction 1 states the order of authority.

  Anthropic: put long documents at the top, each in its own tagged document. S01.

- [ ] **Types, source and stories follow, in that order**

  P02 PROMPT

  Types confirm prop shapes, the implementation shows behaviour the contract leaves out (what `loading` does to clicks), stories supply real usage. Each is its own document.

  Evidence: Documents 2 to 4: `button.types.ts`, `Button.tsx`, `Button.stories.tsx`.

  S01.

- [ ] **Old docs go in last, labelled as possibly stale**

  P03 PROMPT

  Old documentation helps with tone and hurts with facts. It is document 6, marked lowest priority; whatever the contract contradicts becomes a gap.

  Evidence: `--existing` fills document 6; without it the document says None.

  Recommended practice.

- [ ] **Scripts compute what has one right answer**

  P04 PROMPT SCRIPT

  The filler writes the frontmatter facts and the whole API section from the contract. The model copies both and never composes them.

  Evidence: `<frontmatter_facts>` and `<api_section>` in the filled prompt; eval E3 and E4.

  Field Guide 04 W14: generate the API table from the contract.

- [ ] **The model sees every component that exists**

  P05 PROMPT

  Pass the component index with exports and page names, so alternatives, links and the Does not exist list can only name real components.

  Evidence: `design-system.json` is document 5; eval checks E5 and E7.

  Anthropic: restrict the answer to the provided documents. S03.

What the documents do not say, the model will say for them.

SECTION 02

Order it, tag it, explain it.

The filled Button prompt is 33,893 characters. At that length, order and labels matter as much as wording.

Suggested owners: Docs owner

- [ ] **Documents first, instructions last**

  P06 PROMPT

  Long inputs at the top, instructions and output format at the end. Anthropic reports up to 30 percent better responses in its tests when the query comes last.

  Evidence: The template order: documents, page template, facts, API section, examples, instructions, output format.

  S01. The 30 percent figure is Anthropic's, from its own tests.

- [ ] **Every part has its own XML tag**

  P07 PROMPT

  Numbered `<document>` entries with `<source>` metadata, then one tag each for the template, facts, API section, examples, instructions and output format.

  Evidence: `prompt/component-doc.prompt.md`.

  Anthropic: XML tags reduce misinterpretation when a prompt mixes instructions, context and examples. S01.

- [ ] **A role in the system prompt, in one paragraph**

  P08 PROMPT

  The system prompt names the readers (engineers who scan, agents that retrieve one section) and says unsupported facts become gaps. Rules live in the instructions.

  Evidence: `prompt/system.md` is four sentences.

  Anthropic: even a one-sentence role focuses behaviour. S01.

- [ ] **Examples come from other components**

  P09 PROMPT

  Three short examples in `<example>` tags, from Link and Switch. A Button page shown while generating the Button page invites copying instead of reading.

  Evidence: `prompt/examples.md`: three examples, none about Button.

  Anthropic recommends 3 to 5 relevant, diverse examples in example tags. S01.

- [ ] **Instructions say what to do, and why**

  P10 PROMPT

  Each of the 12 instructions carries its reason ("because an agent that only reads don't guesses the alternative"), so the model can apply it to cases the instruction did not list.

  Evidence: Read the `<instructions>` block: every numbered instruction has a because or a reason clause.

  Anthropic: explain why, and say what to do instead of what not to do. S01.

Order, tags and reasons are the prompt. The wording is the least of it.

SECTION 03

Specify the reply so a script can refuse it.

Every instruction about the output maps to a check. What no script can check goes on the reviewer's list in Section 05.

Suggested owners: Docs owner

- [ ] **Gaps are an allowed answer**

  P11 PROMPT

  The prompt gives the model a place to put what it cannot source: a `<gaps>` block of questions for the owner. Without it, the only way to fill a section is to guess.

  Evidence: The sample reply lists four gaps; the eval reports each as a warning.

  Anthropic: allow the model to say it does not know; restrict it to the provided documents. S03.

- [ ] **Structure the reply with tags or a schema, not prefill**

  P12 PROMPT SCRIPT

  Ask for `<page>` and `<gaps>` blocks, or pass `--structured` to get a JSON schema request. Do not prefill the assistant turn: from Claude 4.6 onwards a prefilled last assistant message returns a 400 error.

  Evidence: `fill-prompt.mjs --structured` adds `output_config.format` with `prompt/output.schema.json`.

  Prefill migration and structured outputs. S01, S02. Structured outputs guarantee the shape of the reply, not the truth of the page.

- [ ] **The page is the Field Guide 04 page, exactly**

  P13 PROMPT SCRIPT

  Seventeen H2 sections in the template's wording and order, three accessibility subsections, no placeholders. The template is in the prompt, so the model copies structure.

  Evidence: Eval E2 runs the Field Guide 04 linter; E9 rejects template residue.

  Field Guide 04 W03 and W04.

- [ ] **The API section is copied, never written**

  P14 SCRIPT

  The model reproduces the generated block between the markers. The eval compares it byte for byte, so a helpful rewording fails.

  Evidence: Eval E3 uses the Field Guide 04 `injectApi` comparison.

  Field Guide 04 W14.

- [ ] **Rules carry keywords, reasons and replacements**

  P15 PROMPT SCRIPT

  Three to ten bullets, one RFC keyword each, every MUST NOT and SHOULD NOT with because and use, all taken from the contract.

  Evidence: Linter W10 and W11 through eval E2. The sample has eight rules.

  Field Guide 04 W09 to W12.

An instruction nobody checks is a suggestion.

SECTION 04

Ground every claim, then prove it.

The eval is cheap and exact; a reviewer is expensive and tired. Let the model check itself, then the scripts, before a person reads a line.

Suggested owners: Docs owner + accessibility lead

- [ ] **Examples use only real exports, props and values**

  P16 SCRIPT

  Non-Don't code blocks import only names the index lists and pass only contract props and values. Don't blocks are wrong on purpose and exempt.

  Evidence: Eval E5. On the essay-format page it catches `variant="danger"`.

  Field Guide 02 01.07: closed sets are enums.

- [ ] **The keyboard table is the contract's keyboard list**

  P17 SCRIPT

  One row per key in the contract and no others. A model that knows buttons will add Escape; the contract decides.

  Evidence: Eval E6 compares the rows with `accessibility.keyboard`.

  W3C APG button pattern: Enter and Space. S07.

- [ ] **Behaviour claims are checked by the model before it finishes**

  P18 PROMPT

  The last instruction makes the model reread every sentence about keyboard, focus, announcements and defaults, and turn unsupported ones into gaps.

  Evidence: Instruction 12. The sample reply turned the loading announcement into a gap rather than a claim.

  Anthropic: verify each claim against the documents and retract what has no support. S03.

- [ ] **The eval passes with zero errors**

  P19 SCRIPT

  Run `eval-generated-doc.mjs` on the raw reply. Fix errors by editing the page or regenerating, never by loosening a check.

  Evidence: Sample: first reply 1 error and 4 warnings; edited page 0 and 0 (Appendix B).

  Anthropic: code-graded evals are the automated default. S04.

- [ ] **Examples compile against the real types**

  P20 SCRIPT

  The eval reads examples with a scanner; only the compiler proves they type-check. Extract them with the Field Guide 04 extractor and run `tsc`.

  Evidence: Authoring run: six blocks compiled against the Field Guide 02 types (with stub modules for `@acme/ui`); a `tone="ghost-primary"` control failed as expected.

  Field Guide 04 W19.

The eval decides whether a person reads it.

SECTION 05

Publish only what someone owns.

A passing eval proves the page agrees with the contract. Only people can say it is true and useful, and only a record shows who did.

Suggested owners: Docs owner + component owner

- [ ] **A person checks what no script can**

  P21 REVIEW

  Are the reasons true, is the canonical example the one you want copied, do Variants and content pairs match design intent? The eval proves consistency, not usefulness.

  Evidence: The reviewer's name and date in the record.

  Anthropic Claude Code guidance: a fresh reviewer, and evidence before claims of success. S05.

- [ ] **Every gap is answered before publishing**

  P22 REVIEW

  Gaps go to the component owner. Each answer lands in the contract, stories or source, so the next generation has it too.

  Evidence: Eval E10 lists open gaps as warnings; publish only with none.

  Recommended practice.

- [ ] **The Button is the regression test for the prompt**

  P23 PRACTICE

  When the prompt, examples or model change, regenerate the Button and two other components and diff the pages. A fix for one often breaks another.

  Evidence: Kept replies per prompt version; the eval run on each.

  Anthropic: build evals that mirror the real task distribution, including edge cases. S04.

- [ ] **The prompt is versioned like code**

  P24 PRACTICE

  Keep the template, examples and filler in the design-system repo, and record the prompt commit and model id with each page.

  Evidence: The review record at the end of this guide.

  Recommended practice.

- [ ] **The generated page is a draft until someone owns it**

  P25 REVIEW PRACTICE

  The frontmatter says `method: manual-review`, which is true only after a named person reviews it. Until then the page stays on a branch.

  Evidence: A published page has a reviewer in its record and no open gaps.

  Field Guide 21 scores doc drafting as assist (draft).

The person decides whether anyone else reads it.

APPENDIX A

The prompt, as a file with variables.

Eighteen `{{VARIABLES}}` across the prompt and the system prompt, all filled by the script. An unknown or unfilled variable is an error, never a blank.

prompt/component-doc.prompt.md (abbreviated)

```
<documents>
<document index="1">
<source>{{CONTRACT_PATH}} (component contract, Field Guide 02: the source of truth)</source>
<document_content>{{CONTRACT}}</document_content>
</document>
  ... 2 to 6: types, implementation, stories, component index, existing docs
</documents>

<page_template>{{PAGE_TEMPLATE}}</page_template>
<frontmatter_facts>{{FRONTMATTER_FACTS}}</frontmatter_facts>
<api_section>{{API_SECTION}}</api_section>
<examples>{{EXAMPLES}}</examples>

<instructions>
Write the documentation page for {{COMPONENT_NAME}} from {{PACKAGE}} {{PACKAGE_VERSION}}. ...
1. Use only the documents above. Their order of authority is: contract, prop types, ...
4. Copy the API section exactly as given ... a CI check compares it byte for byte.
12. Before you finish, check every sentence that describes behaviour ... add a gap.
</instructions>

<output_format>
Reply with two blocks and nothing else: the page in <page></page>, then <gaps></gaps> ...
</output_format>
```

The full file has twelve numbered instructions, each with its reason. The system prompt is `prompt/system.md`.

Variable	Filled from
`CONTRACT`, `TYPES`, `SOURCE`, `STORIES`	The component folder: `*.contract.json`, `*.types.ts`, the `.tsx` file, `*.stories.tsx`
`COMPONENT_INDEX`	`design-system.json`: packages, exports and page names
`EXISTING_DOCS`	`--existing page.md`, or None
`PAGE_TEMPLATE`	The Field Guide 04 template
`FRONTMATTER_FACTS`	Computed from the contract and the index
`API_SECTION`	Field Guide 04 `apiTable()` over the contract, with markers
`EXAMPLES`	`prompt/examples.md`

APPENDIX B

One run, from folder to passing page.

Output below is from the kit. The reply in `examples/button.reply.md` is the untouched first generation; `examples/button.generated.md` is the page after one hand edit.

terminal

```
$ node scripts/fill-prompt.mjs examples/button --out build --today 2026-09-24
Wrote system.md, prompt.md and request.json to build
button.contract.json: 33893 characters (roughly 8473 tokens at 4 characters per token; ...)

$ node scripts/eval-generated-doc.mjs examples/button.reply.md \
    --contract examples/button/button.contract.json
error  E8 "Does not exist" names Button, which exists
warn   E10 open gap: Accessibility: ... Does Button announce when loading ends, ...
warn   E10 open gap: States: the contract lists an active state but no token ...
warn   E10 open gap: Tokens: the contract lists hover tokens only for the primary tone ...
warn   E10 open gap: Do and don't: ... Which other props and components have evals caught?
button.reply.md: 1 error(s), 4 warning(s)

$ node scripts/eval-generated-doc.mjs examples/button.generated.md \
    --contract examples/button/button.contract.json
button.generated.md: 0 error(s), 0 warning(s)

$ node scripts/eval-generated-doc.mjs examples/button.essay-format.md \
    --contract examples/button/button.contract.json
error  E2 lint: page: missing YAML frontmatter
error  E3 Missing <!-- api:start --> ... <!-- api:end --> markers
error  E4 no frontmatter to compare with the contract
error  E5 line 29: <Button> has no prop variant
error  E6 keyboard table is missing Enter, Space, Tab from the contract
button.essay-format.md: 5 error(s), 0 warning(s)
```

Gap text shortened for print. `button.essay-format.md` is written by hand to the output format of the essay's original prompt. The eval catches its invented `variant` and missing keyboard table; its extra Escape key and `role="button"` need a reviewer. The linter stops at missing frontmatter, so it has more problems than five.

What was edited by hand.

One sentence. The first reply wrote "the only button component is `Button`" inside the Does not exist paragraph, and E8 rejected a real name in a list of names that do not exist. The edit moved Button out of that sentence. Nothing else changed; the four gaps stay open for the owner and live in the reply, not the page.

The sample was generated by the same model family that wrote this kit, which had also read the Field Guide 04 Button page. Treat it as a demonstration of the pipeline, not as evidence of how any model performs. The reply predates the reason clauses in instructions 1, 2, 3, 5, 8, 9, 10, 11 and 12; what each instruction asks for did not change.

APPENDIX C

Ten checks, and what they leave to you.

The eval imports the Field Guide 04 linter and API generator and the Field Guide 20 scanner, so the three guides share one implementation of each rule.

Check	What it proves	Severity
E1	The reply contains a page (tags, JSON or bare Markdown)	error
E2	The Field Guide 04 linter passes: structure, rules, fences, wording	lint severity
E3	The API section equals a fresh generation from the contract	error
E4	Frontmatter title, package, import, status and pattern link match the contract	error
E5	Examples import real exports and pass only contract props and values	error
E6	The keyboard table equals the contract's keyboard list	error
E7	Component links point at pages in the component index	error
E8	The Does not exist list names nothing that exists	error
E9	No unfilled variables, template placeholders, TODOs or leftover comments	error
E10	Open gaps, listed for the owner	warning

DO

- Check that each reason in Rules is the real reason
- Check that the canonical example is the one you want copied
- Use the component: states, focus, announcements
- Answer every gap in the contract, stories or source

DON'T

- Edit the API table by hand
- Loosen a check to pass a page
- Publish with open gaps
- Show the model a finished page of the same component as an example

KEEP WITH THE PAGE

Leave a generation record.

One record per generated page. It lets the next person trace a wrong sentence to the prompt, the model or the reviewer.

Component / contract version	**Prompt commit / examples commit**
Model id and date	**Filled prompt size (characters)**
Eval result on the raw reply	**Hand edits (what and why)**
Gaps and who answered them	**Examples compiled (link)**
Reviewer (name, date)	**Published page URL**

> **Keep the raw reply.**
> The reply, not the edited page, is what tells you whether the prompt is getting better.

SOURCES / MAINTENANCE

Keep the guide current.

Sources checked 24 September 2026. Anthropic's figures come from its own tests; treat them as signals. The essay is cited for the prompt this guide replaces.

S01 / Anthropic, prompting best practices
Long documents first, document and source tags, 3 to 5 examples in example tags, roles, reasons, and migrating away from prefill on Claude 4.6 and later.
https://platform.claude.com/docs/en/build-with-claude/prompt-engineering/claude-prompting-best-practices

S02 / Anthropic, structured outputs
output_config.format with a JSON schema; additionalProperties false; no string length or pattern constraints.
https://platform.claude.com/docs/en/build-with-claude/structured-outputs

S03 / Anthropic, reduce hallucinations
Allow uncertainty, restrict to provided documents, verify claims against quotes and retract unsupported ones.
https://platform.claude.com/docs/en/test-and-evaluate/strengthen-guardrails/reduce-hallucinations

S04 / Anthropic, define success criteria and build evaluations
Task-specific evals, automated grading where possible, code-graded checks.
https://platform.claude.com/docs/en/test-and-evaluate/develop-tests

S05 / Claude Code, best practices
Give the model a way to verify its work; review in a fresh context; show evidence rather than assert success.
https://code.claude.com/docs/en/best-practices

S06 / Petri Lahdelma, AI in design systems: what actually works
The essay whose Steal this list promised this prompt, and its first version of it.
https://petrilahdelma.com/writing/ai-design-systems-practical

S07 / W3C, ARIA Authoring Practices: button pattern
Enter and Space activate a button; the source for the keyboard table.
https://www.w3.org/WAI/ARIA/apg/patterns/button/

Maintenance: recheck Anthropic's prompting and structured output pages with each new model generation (prefill support changed in 2026), and rerun the Button regression after every prompt edit. Update the PDF, HTML, Markdown and JSON together.