MD2FILE RESEARCH · EDITION 1.0.1

The Markdown Layer

State of Markdown 2026

Repository evidence, agent instructions and document workflows.



An original 24-repository panel, six focused research chapters, reproducible data and a clearly labeled 2027 outlook.

MD2FILE Team
First published September 30, 2026
Editorial revision October 15, 2026
Research evidence reviewed September 30, 2026
md2file.com/blog/the-markdown-layer/

Contents

This report combines a concise overview with chapters that answer different questions. Definitions recur where a chapter also stands alone on the website. The canonical web hub links to each chapter and the underlying data.

1. Executive overview
2. Markdown in GitHub Repositories: A 2026 Snapshot
3. Markdown for AI agents: what instruction files can and cannot do
4. Maintaining AI agent documentation without conflicting instructions
5. Markdown as a Knowledge Format: What Survives Between Tools?
6. How to Check a Markdown PDF Before Sharing
7. AI Conversations as Documents: Preserve Context, Sources and Decisions
8. The Markdown Layer: Methodology and Research Data
9. File concentration and the 2027 research agenda
10. Source register and editorial disclosure

How to read the numbers

Keep the population, observation date and measurement rule with every count. Published studies and the MD2FILE panel use different selection rules. Our panel has 24 selected repositories, 21 complete trees and three unknown trees. It does not estimate adoption across GitHub.

Disclosure

MD2FILE publishes this report and sells the tools discussed in it. AI assisted research, drafting, code and the cover illustration. Statistical charts come from retained data; workflow diagrams are conceptual. This is not a user survey, product security audit or causal SEO study.

CHAPTER 1

Executive overview

Repository evidence, agent instructions and document workflows.

Markdown now serves several jobs in the same project: explaining software to people, holding instructions for coding agents, and supplying source text for published documents. The evidence is strongest for its presence in established public repositories. It is much weaker on how often people move that text between tools, whether instruction files improve an agent's work, or how much demand exists for document export.

The Markdown Layer examines those boundaries. This edition combines published research, current format and product documentation, and a small original repository study. It offers a practical model for keeping source text understandable as it moves between people, software and AI systems.

Published: September 30, 2026. **Edition:** 1.0.1, October 15, 2026. **Publisher:** MD2FILE Team. This revision clarifies Chrome and Safari support in the AI conversations chapter. The research evidence and repository panel retain their September 30 review date. The closing 2027 outlook is a set of questions to test, not a report of future results.

What the evidence supports

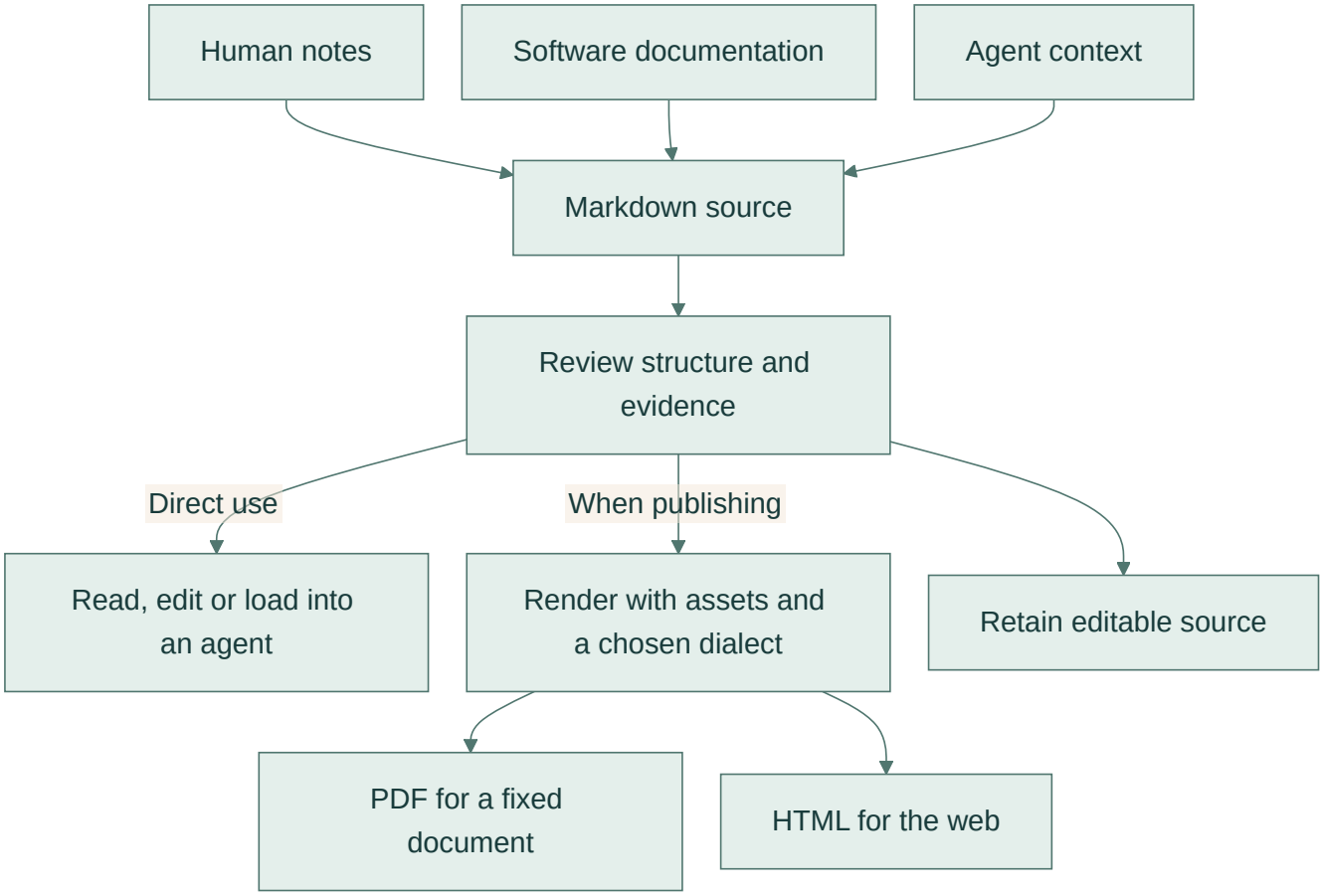
Five findings shape this report:

1. **Markdown is well established in the public software projects studied.** A 2026 academic sample found `README.md` somewhere in 9,532 of 10,000 selected repositories. That population had activity, star and commit requirements; it does not represent every GitHub repository.
2. **Agent instruction files have become a measurable repository artifact.** A separate September census found at least one recognized instruction filename in 2,463 of 7,370 large, recently active repositories. Presence alone says nothing about whether an agent used the file successfully.
3. **Our original panel exposes coexistence and uneven volume.** We inspected 21 complete trees from 24 selected repositories. All 21 contained Markdown; 15 contained at least one of four context filename candidates. Two repositories accounted for 27,715 of the panel's 35,904 Markdown files.
4. **Agent effectiveness remains conditional.** Experiments use different tasks, agents, instruction sources and outcomes. A reduction in runtime or output tokens does not establish better correctness, lower total token use, or a benefit for another repository.
5. **Portable text still needs a rendering and provenance check.** A file can survive transfer while its diagrams, relative images, citations or page layout fail. Keeping the source and documenting the export decision helps the next reader understand what they received.

The first two figures come from Hora and colleagues' repository study and Stride's September census. Their populations and methods differ. Our original panel and limitations are published separately so the three datasets remain distinguishable.

One format, several responsibilities

Markdown's original design favored source that a person could read before conversion to HTML. That remains useful when the same text appears in a repository, editor, note system or prompt. Its readable syntax lowers one transfer cost: a recipient can often inspect the content without the application that created it. It does not carry every application's behavior with it. See the original Markdown description and the CommonMark specification.



Conceptual Markdown lifecycle: reviewed source can be read, edited or loaded into an agent, with a separate publishing path to PDF or HTML and editable source retained.

Consider a team preparing a release. Its README explains installation, a scoped instruction file tells an agent which test suite to run, and a release note records the changes. A PDF version might go to a customer who does not use the repository. These artifacts can share a syntax while requiring different checks. The installation command must work, the instruction must apply to the current directory, and the customer document must remain readable without repository-relative assets.

We use **Markdown layer** as an editorial description of that shared text surface. It is not a new standard or a claim that every AI system uses Markdown internally. The useful question is what must accompany the text at each handoff: context, permissions, assets, review history or renderer settings.

Handoff	What the text carries	What still needs checking
Maintainer to contributor	Explanation, commands, links	Commands and links match the current project
Maintainer to coding agent	Task context and repository conventions	Scope, precedence, actual tool permissions
Writer to another editor	Headings, paragraphs, source references	Dialect, attachments, application-specific features
Editor to reader	A rendered document	Fonts, diagrams, pagination and provenance

CHAPTER 2

Markdown in GitHub Repositories: A 2026 Snapshot

All 21 complete repository trees in MD2FILE's September 30, 2026 snapshot contain Markdown and a root Markdown README. Fifteen contain at least one recognized agent-context filename. We selected 24 repositories; three file trees exceeded our collection limit and remain unknown. These counts describe a small panel selected by GitHub stars, not Markdown adoption across GitHub.

The repository dataset records the exact selection, commit identifiers and missing coverage. This study is part of The Markdown Layer, which examines how structured text moves between documentation, software and AI workflows. MD2FILE publishes the research and sells document tools; no repository in the panel was selected for using an MD2FILE product.

What can a repository tell us about Markdown use?

A repository tree can establish that a file exists at a particular commit. It can show the file's name, location and stored size. Those observations support questions about documentation structure: whether a project has a README, whether instructions appear in several directories, and whether multiple instruction-file conventions coexist.

The same tree cannot establish how often someone reads a document, whether an agent loads it, or whether it helps complete a task. A large collection of Markdown files may be a course, translated documentation, generated material or test data. A filename count needs a definition before it becomes a useful statistic.

Our study records regular tracked files with `.md` or `.markdown` extensions, ignoring extension case. MDX is separate because it permits embedded components and belongs to a different rendering contract. We also record several exact context-file names, without assuming that every matching file is active configuration.

How we selected the 24 repositories

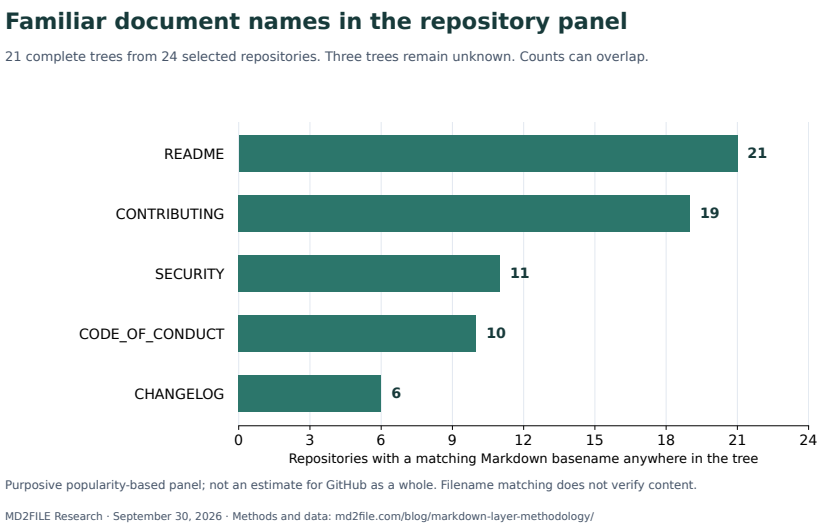
We ran four public GitHub repository searches, using Python, TypeScript, Rust and Go as language filters. Each query required at least 1,000 stars, a push on or after January 1, 2026, and a public repository that was neither a fork nor archived. We retained the first six results sorted by descending star count, including repositories whose names or purposes did not fit a conventional application project.

This selection favors highly visible projects. It also includes curated lists and coding-agent software, both of which can affect the results. GitHub's language metadata describes repositories; it does not identify the languages spoken by their contributors or every technology they use. The exact queries and reproduction procedure make those choices inspectable.

Collection ran on September 30 between 13:52 and 13:55 UTC. Each default branch was resolved once to an immutable commit and tree identifier. We retrieved 21 complete trees. The responses for `openclaw/openclaw`, `rust-lang/rust` and `microsoft/TypeScript` exceeded our 12 MiB client limit. We kept all three in the selected panel with unknown counts and made no replacement selection.

README and contribution files remain visible beside agent files

The 21 complete trees contain Markdown files serving several recognizable naming conventions. A repository can contribute to several rows, and most rows allow matches anywhere in its tree.



Counts of document filenames in the 21 complete repository trees; the three incomplete trees are excluded from these counts

Observed filename or directory	Complete repositories containing it
Root `README.md` or `README.markdown`	21 of 21
`CONTRIBUTING.md` or `.markdown`, anywhere	19 of 21
`SECURITY.md` or `.markdown`, anywhere	11 of 21
`CODE_OF_CONDUCT.md` or `.markdown`, anywhere	10 of 21
`CHANGELOG.md` or `.markdown`, anywhere	6 of 21
Exact root `docs/` directory	11 of 21
A recognized context-name candidate	15 of 21

Source: processed panel, September 30, 2026. Human-document basenames are case-insensitive. Context-name matching follows the stricter rules below. The counts omit three unknown trees and must not be presented as 24 successful inspections.

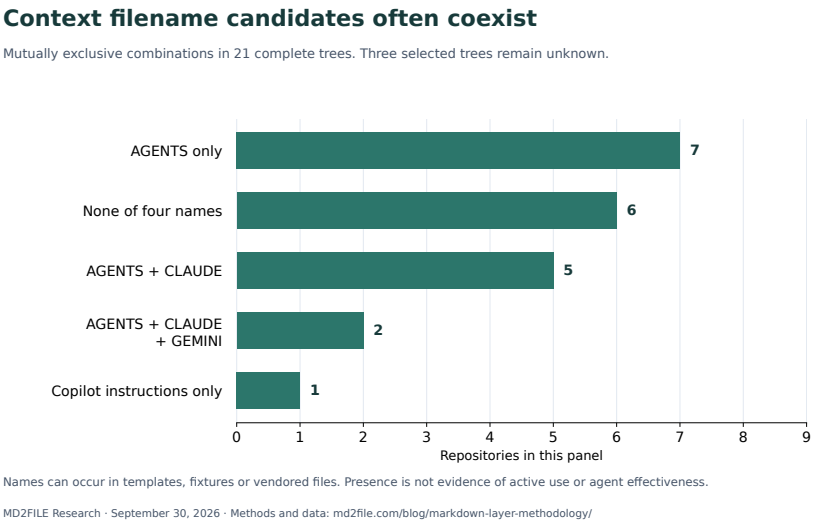
A project without the exact `docs/` directory can still have extensive documentation elsewhere. Likewise, release notes can live in GitHub releases, a differently named file or another website. Absence from one filename rule is narrower than absence of the underlying practice.

For practical README authoring, the existing GitHub-flavored Markdown guide includes a worked source file and its PDF. That tutorial addresses export behavior; this repository study did not render or test third-party documents.

Context-file conventions overlap

The main context-name rule recognizes case-sensitive `AGENTS.md`, `CLAUDE.md` and `GEMINI.md` basenames anywhere in the tree, plus the exact path `.github/copilot-instructions.md`. Among the 21 complete trees, 14 contain an `AGENTS.md` candidate, seven contain `CLAUDE.md`, two contain `GEMINI.md`, and one contains the Copilot path.

Seven repositories contain more than one of these categories. Nine contain a recognized candidate below the repository root. These observations make coexistence and directory scope useful questions for documentation maintainers. They do not show whether the instructions agree or which one a particular agent reads.



Mutually exclusive context filename combinations in 21 complete trees; three selected trees remain unknown

The paths expose reasons for caution. Kubernetes contains named candidates inside `vendor/`; n8n includes candidate files inside a template directory. Other projects have instructions near tests. A name match in one of these locations may describe a dependency, an example or a directory-specific workflow. The classified path CSV retains those locations instead of silently treating every match as a repository-wide policy.

`SKILL.md`, GitHub instruction files and Cursor `.mdc` files are separate fields in the dataset. They are not added to the headline context count. The chapter on AI-agent instructions explains why loading rules matter, while maintaining agent documentation addresses ownership and conflicting instructions.

Why a total file count can mislead

The 21 complete trees contain 35,904 Markdown files. Two repositories account for 27,715 of them: `freeCodeCamp/freeCodeCamp` has 16,810, and `nilbuild/developer-roadmap` has 10,905. Their size has far more influence on the total than a small project with a handful of Markdown documents.

Those files total 152,448,261 stored Git-blob bytes across the complete panel. Bytes measure source size, including syntax and whatever content the files hold. They are not words read, documents exported or evidence of Markdown's share of all software work.

For that reason, we emphasize repository-level presence and provide the per-repository counts. A researcher interested in documentation volume would need to separate generated content, translations, educational material and duplicated examples. This snapshot makes no such content classification.

How this relates to larger research

Hora, Montandon and Costa's repository-content study, version 2 reports `.md` files in 9,860 of 10,000 sampled repositories and `README.md` in 9,532. Their sampling frame contains 116,013 non-fork repositories active in 2026 with at least 100 stars and 100 commits. Those eligibility rules and their random selection differ substantially from our small popularity-based panel.

The paper's historical slices use repositories from the 2026 sample that existed in earlier years. Changes across those slices should be read with that surviving-sample limitation. Our own snapshot supplies no historical comparison at all. Combining the two datasets into a single adoption rate would obscure their different populations and file definitions.

The larger study supports discussion of documentation patterns within its stated sample. Our panel supplies a smaller, inspectable set of current paths and immutable references. Neither dataset measures all private repositories, local notes, AI conversations or the behavior of all developers.

What another snapshot should preserve

A useful follow-up can revisit the same repository identities at new pinned commits on a later date and record additions, removals and renamed files. It should retain the three current unknowns as missing baseline observations. Substituting new popular repositories would answer a different question about a changing leaderboard.

Readers can download the repository CSV and reproduction script. The methodology explains the byte ceiling, matching rules, API request counts and limits on reuse. A correction should identify the repository and commit so that another reader can inspect the same evidence.

CHAPTER 3

Markdown for AI agents: what instruction files can and cannot do

Markdown instruction files give coding agents persistent project guidance. They can describe a required test command or a convention that would otherwise need explaining in each task. Research documents their adoption, but does not establish that adding an `AGENTS.md` file generally improves correctness or reduces costs. The contents, agent and task all need consideration.

This chapter of The Markdown Layer separates three questions: whether a file exists, whether an agent loads it, and whether its instructions help. Those are different measurements.

What is an agent instruction file?

An agent instruction file is a text artifact intended to guide a coding agent's work in a project. The AGENTS.md format project describes a shared location for guidance such as build steps, tests and coding conventions. Markdown provides headings, lists and code blocks that a maintainer can read and edit alongside the code.

A repository might need to explain that a generated directory must be updated through its generator, or that a fast test suite excludes integration tests. Those statements supply working context. They also create maintenance obligations: when a command or directory changes, the instruction may need to change with it.

Artifact	Question it should answer	Example content
Human README	What is this project and how do I start?	Purpose, installation, a working example
Agent instruction file	What must an agent know while changing this project?	Required checks, scoped conventions, generated-file boundaries
Task description	What should change in this request?	Expected behavior and acceptance criteria
Historical decision record	Why was a choice made?	Alternatives considered and the dated decision

These are editorial roles, not a universal directory structure. A small project can keep related material together. The practical test is whether a reader can identify which statement governs the current task.

How common are these files?

Stride Research reported instruction files in 2,463 of 7,370 eligible repositories on September 27, 2026, or 33.4%. Its population was public, non-fork, non-archived GitHub repositories with at least 5,000 stars and a push in the preceding 90 days. Root `AGENTS.md`, including filename case variants, appeared in 25.3%; root or `.claude/CLAUDE.md` in 19.2%. A repository could contain both. Study and methodology.

MD2FILE recomputed those counts from the published CSV. That checks the reported arithmetic, not the completeness of the GitHub census or whether every detected file was used. Stride is a commercial publisher, and its population excludes smaller and private repositories. The repository adoption chapter explains why percentages from different samples should not be combined into a global adoption rate.

Do AGENTS.md files improve agent performance?

The answer depends on which outcome is measured. A fast run can produce an incorrect patch; a correct run can spend more time checking constraints. The two studies below therefore need to remain separate.

Study	Evaluation	Reported result	Limit
Lulla and colleagues, March 2026 revision	124 small pull-request tasks from 10 repositories; paired Codex runs with and without the existing root file	Median runtime fell 28.64%; median output tokens fell 16.58%	Comprehensive correctness evaluation was outside scope
Gloaguen and colleagues, September 29 revision	SWE-bench Lite and CTXbench; multiple model/harness setups, with absent, generated or developer-committed context	No statistically significant success improvement against absent context; generated files increased average inference cost	Python benchmark tasks and the tested configurations

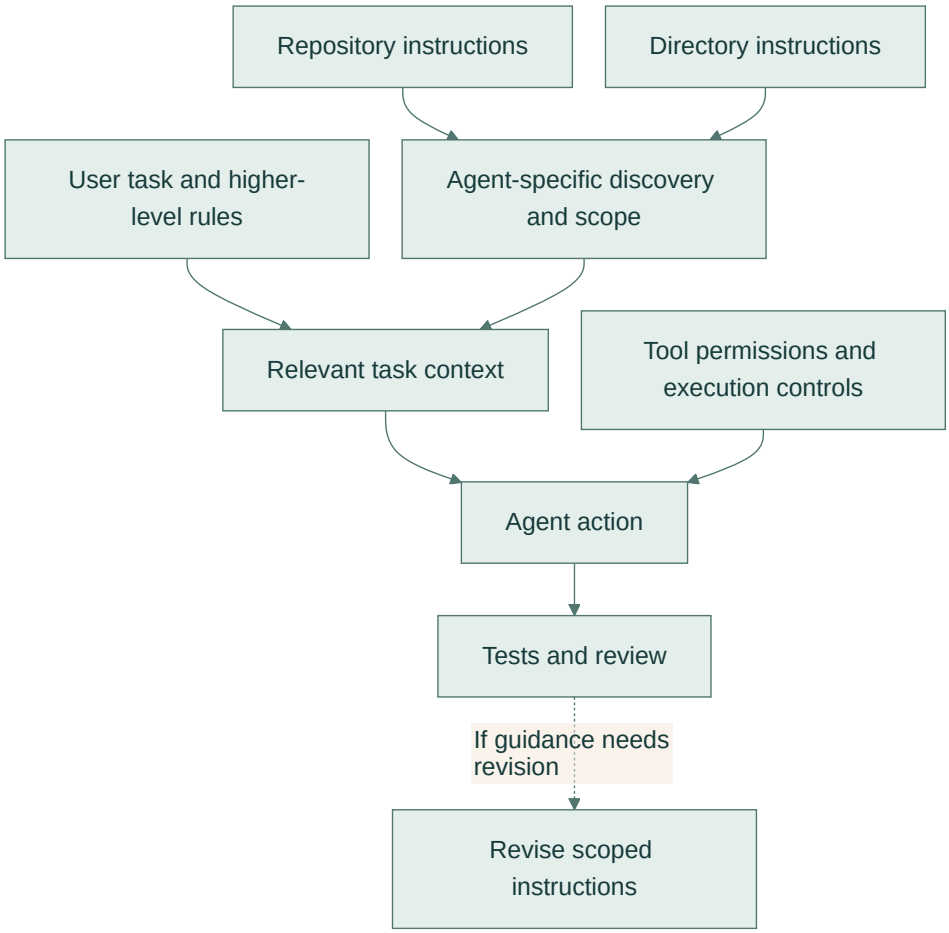
The efficiency study used `gpt-5.2-codex` and manually sanity-checked outputs for 50 tasks. Its token reduction concerns **output tokens**; median total tokens did not fall. It provides a reason to investigate efficiency in a similar workflow, not a promise of lower bills or equivalent code quality.

The revised effectiveness study used 300 SWE-bench Lite tasks and 138 CTXbench tasks. Generated context increased average costs by 20% and 23% respectively. Developer-committed files performed better than generated files, but did not significantly outperform having no file. A nonsignificant difference does not prove that every instruction is useless.

These results do not supply a universal word limit or an ideal template. They suggest that adopting the filename and demonstrating an improvement are separate steps. A team still needs to decide what result would justify keeping a particular instruction.

A file must be loaded before it can influence a task

Discovery rules depend on the agent and version. For example, Anthropic's current documentation says Claude Code can read `AGENTS.md` directly from version 2.1.277, with default behavior affected by `CLAUDE.md` files in the project hierarchy. Imports and settings can change that behavior. Verify the actual loaded context before diagnosing an ignored instruction. Claude Code documentation.



Conceptual flow from discovered repository instructions, the user task and higher-level rules to task context, actions, tests and review. Tool permissions separately constrain execution; review leads to revised instructions when a guidance problem is found.

Conceptual diagram by MD2FILE. It separates guidance, loading and execution controls; it is not a precedence specification for every agent.

Keep permissions explicit outside the prose. Anthropic describes instruction files as context rather than enforced configuration. A sentence asking an agent to avoid an action should not be the only control preventing that action. Review the tool's permission settings separately from the wording of its instructions. Memory and instruction behavior.

Write an instruction that can be checked

Consider a fictional package whose client code is generated from an API schema. “Keep the client consistent” leaves the expected work unclear. A more useful draft names the source and the verification step:

```
## Generated API client

Edit api/schema.yaml when changing an endpoint.
Regenerate packages/client with the project's client generator.
Include the schema and generated-client diff in the same review.
Do not describe the change as verified if generation fails.
```

This example deliberately avoids inventing a command for your repository. Replace the generator reference with a command that actually exists, and run it before publishing the instruction. If regeneration changes unrelated files, investigate that difference instead of teaching the agent to accept it automatically.

The instruction has a defined subject, scope and observable result. A reviewer can check whether the schema changed, whether generation ran and whether the resulting diff belongs to the task. It is still possible for an agent to misunderstand or disregard it, so those checks remain necessary.

Avoid adding a permanent rule for every unusual incident. A temporary migration exception can belong in the migration task. A recurring constraint on all client changes belongs in maintained project guidance. This choice reduces the chance that an old exception will be mistaken for a current requirement.

Evaluate one change to the instructions

Use a small set of representative tasks before expanding the file. Include an ordinary change and a case where the proposed instruction should matter, such as modifying the schema in the example above. Keep the repository revision, agent version, model and permissions the same across comparison runs.

Record each outcome separately:

- Did the patch satisfy the task and pass the relevant tests?
- Did it respect the particular requirement being evaluated?
- What were the elapsed time, input tokens, output tokens and charged cost, where available?
- How much correction or review did a person need to perform?

Repeat tasks where practical and retain failed runs. Report the number of trials and variation, rather than selecting the best result. A single successful demonstration can show that a workflow is possible; it cannot establish a reliable average improvement.

Keep a useful instruction with its evidence and owner. Revise an ambiguous instruction, and remove one whose purpose has expired. The companion chapter on maintaining agent documentation develops that review process, including how to separate current requirements from archived decisions.

Evidence and publisher disclosure

MD2FILE publishes this research series and provides Markdown editing and conversion tools. The cited studies evaluate repository practices and coding agents, not MD2FILE products. We did not rerun their agent experiments. Source versions and sampling limits are recorded in the series methodology.

Previous: Repository adoption evidence · Overview: The Markdown Layer · Next: Maintaining agent documentation

---

CHAPTER 4

Maintaining AI agent documentation without conflicting instructions

Maintain agent documentation by giving each current instruction a clear scope, an owner and a way to verify it. Separate working requirements from historical notes, and review affected guidance when the code changes. The process below is a proposed maintenance practice, not a measured productivity improvement.

The Markdown Layer research overview describes Markdown's use across documentation and agent workflows. This chapter addresses a narrower problem: what to do when several readable files give incompatible advice about the same task.

What does the research establish about documentation sprawl?

Harsha Kokel's *Markdown Mayhem*, an IBM-affiliated workshop position paper published in May 2026, argues that agent documentation needs clearer authority and governance. It discusses ambiguity and redundant instructions as risks. It does not measure a worldwide rate of documentation growth or prove that a particular maintenance process prevents failures. Author's paper.

There is empirical evidence that configuration artifacts are present in repositories. Galster and colleagues' published study found context files in 2,586 of 2,853 repositories with detected agent configuration. Those repositories came from a filtered set of established GitHub projects, with an artifact snapshot in February 2026. The 90.6% figure describes detected adopters, not all repositories or all developers. Published study.

Neither observation tells a maintainer whether their own instructions conflict. That requires examining the relevant files and the behavior they request. The number of Markdown files alone is a poor target for a cleanup: a large reference library can be coherent, while two short files can contradict each other.

Start with a concrete contradiction

Suppose a fictional repository contains these statements:

Location	Statement	What a maintainer needs to resolve
Root agent instructions	Run the quick test suite before submitting changes	Whether quick tests cover this task
API package guide	Run the contract suite for public API changes	Whether this adds a scoped requirement
Old migration note	Skip contract tests while the test service is unavailable	Whether the temporary exception still applies

The first two requirements may be compatible. The third needs evidence about its status. Deleting the longer file or telling the agent to prefer the newest timestamp would not resolve the underlying question.

Check the current test service, the package's test configuration and the decision that introduced the exception. If the exception has ended, update the live guidance and mark the migration note as historical. Preserve the reason for the original decision where it helps explain the migration, with a link to the replacement requirement.

An unresolved conflict should stay visible to the reviewer. Do not silently rewrite an uncertain policy into a confident instruction merely to make the files agree.

Inventory the instructions an agent can actually receive

List the entry files, nested instructions and referenced documents used by the team's agent setup. Record their audience and scope. A general search for Markdown is useful for discovery, but it does not tell you which files a particular runtime automatically loads.

Loading behavior can also change after an upgrade. Anthropic's documentation, for example, describes different treatment of project files, imports and scoped rules. Check the documented behavior for the version you use, then inspect the session's loaded context where the tool exposes it. Claude Code memory documentation.

Record	Why keep it?
File path and intended reader	Distinguishes current agent guidance from human reference material
Owner or reviewing team	Identifies who can resolve a disputed requirement
Applicable package or task	Prevents a local exception from becoming a project-wide rule

Record	Why keep it?
Source of the requirement	Connects prose to a configuration, decision or tested workflow
Last verification and trigger for review	Explains what was checked and what might invalidate it

A review date should describe an actual check. Changing a date without verifying the content makes the record less useful. If only links were checked, say so; that does not establish that a deployment procedure or test command still works.

Give recurring requirements one maintained home

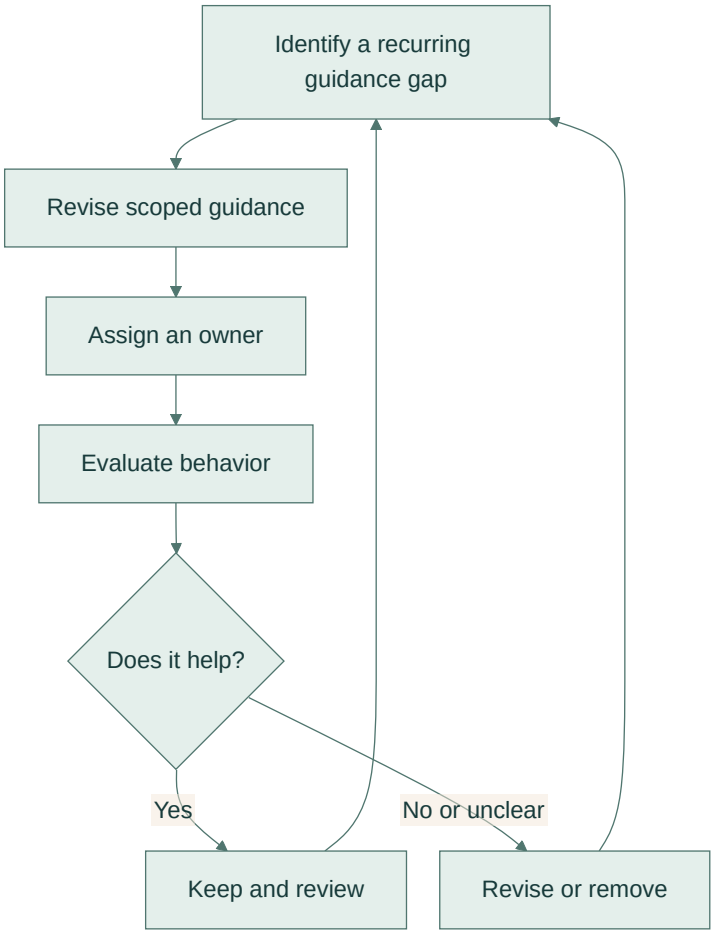
Choose one authoritative source for each repeated requirement. A test command can live beside the package configuration or in a maintained contributor guide. Agent instructions can point to that source, provided the workflow actually brings the relevant material into context when needed.

Copying the same command into several files creates several places to update. A link reduces that duplication, but introduces a different failure: the reader may never follow it. Decide whether the instruction must be immediately available or can be consulted for a particular task, and verify that behavior in the chosen tool.

Avoid collapsing every document into one large file. Keep detailed migration history where a maintainer can find it, while making its status unambiguous. The knowledge portability chapter discusses the wider problem of preserving meaning when a document moves between tools.

Review documentation with the code change

Attach documentation review to a relevant event. Renaming a command should trigger a search for that command in instructions and examples. Moving a package should trigger a check of scoped paths. Changing an export format should trigger a check of sample output and the instructions used to produce it.



Proposed maintenance workflow: identify a recurring guidance gap, revise scoped instructions, name an owner, evaluate the guidance, then keep, revise or remove it and revisit it when code changes.

MD2FILE's proposed review cycle. It is a conceptual workflow, not an experimentally validated intervention.

For a command change, a useful review record might look like this:

```
## Instruction review: API tests

Reason: the contract-test command changed in this revision.
Scope: public API changes in packages/api.
Owner: API maintainers.
Checked: the documented command ran in the supported test environment.
Updated: the package guide and its agent-facing reference.
Historical note: the migration exception is marked superseded.
Open issue: Windows instructions have not been verified.
```

This is an illustrative record. It makes no claim that these checks have been performed in your project. Adapt the fields to the decision being made and leave unverified cases explicit.

Check behavior as well as the document

A clean document can still describe an ineffective workflow. After changing an instruction, try a task that should exercise it and inspect both the result and the agent's actions. If the instruction asks for a contract test, confirm that the relevant test ran and that its result was handled correctly.

Compare the changed instruction against the previous version when the behavior is important enough to justify the work. Use the same task and repository snapshot, and record agent/model versions. Assess task correctness separately from duration and token use; the agent-instruction evidence review explains why those outcomes can disagree.

Some requirements express policy rather than an efficiency goal. A rule preventing an unauthorized release should not be removed because it slows a benchmark. Define the intended outcome before the trial, and use permissions or other execution controls for boundaries that must be enforced.

Archive instructions without losing their explanation

When guidance expires, remove it from active entry points and preserve useful history with a clear status. An archive record should identify what superseded it and when. If a historical page remains accessible, its opening should make that status understandable without requiring a reader to find another file first.

Keep the editable source when sharing a fixed review copy. A PDF can be useful for a meeting or approval record, but it needs a revision identifier and a link to the maintained source so readers can distinguish that snapshot from current instructions. The document rendering chapter covers the checks needed when publishing such a copy.

Evidence and publisher disclosure

MD2FILE publishes this series and provides Markdown editing and conversion tools. Its commercial interest does not establish a need for more instruction files or a particular publishing format. The maintenance examples above are proposals; we have not measured their effect on agent reliability. The series methodology records research versions, sampling limits and verification scope.

Previous: Markdown for AI agents · Overview: The Markdown Layer · Next: Knowledge portability

---

CHAPTER 5

Markdown as a Knowledge Format: What Survives Between Tools?

Markdown makes written knowledge easy to move as text. A note can remain readable outside the application that created it, and its headings, links and code can be inspected without opening a proprietary document format. Moving the file, however, does not necessarily preserve its attachments, application features or rendered appearance.

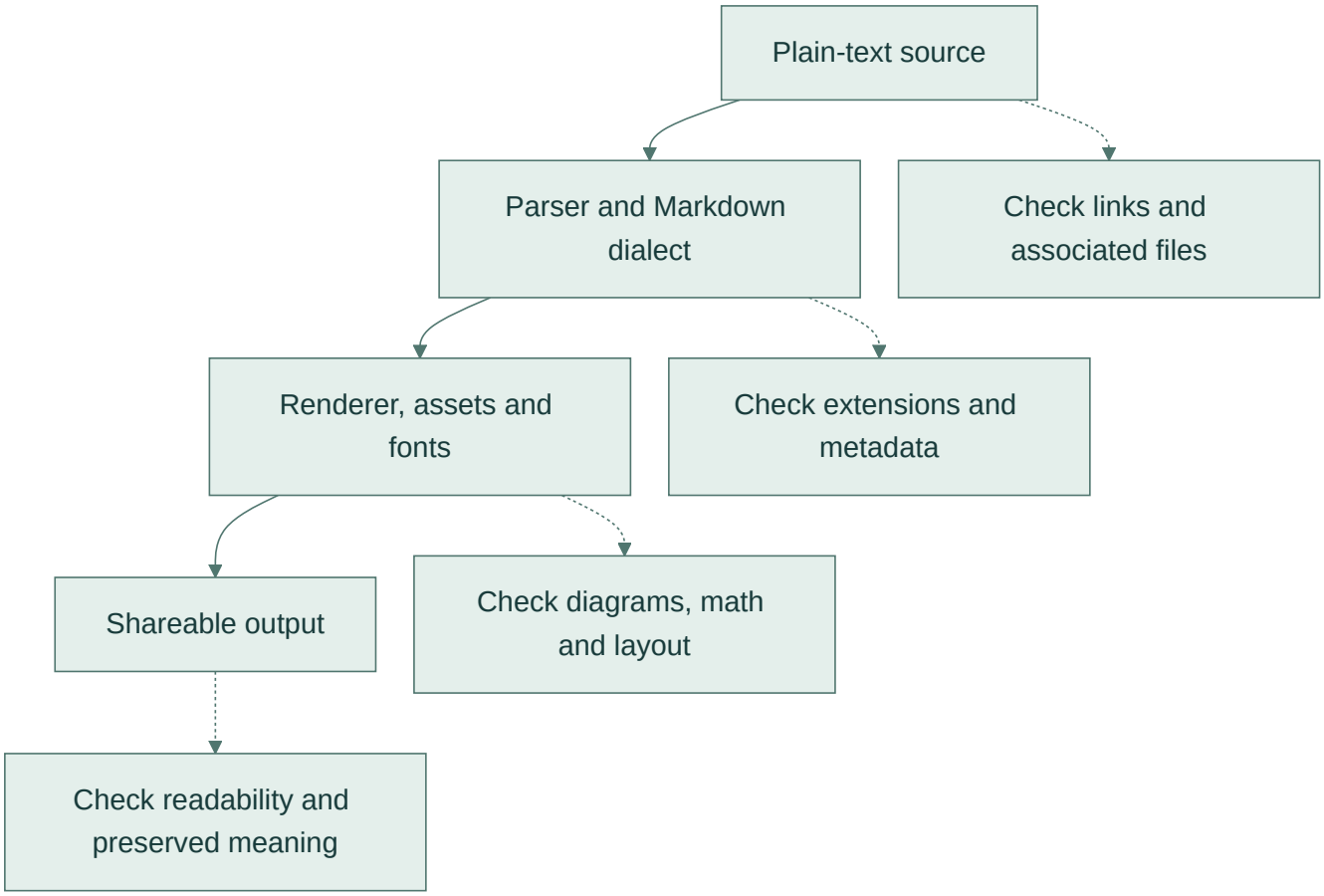
That distinction matters when a personal note becomes team documentation or a repository README becomes a client report. A useful portability check asks what the next reader needs to understand, then checks whether the exported package supplies it. The review method below is editorial guidance, not a measured comparison of knowledge tools.

This chapter is part of The Markdown Layer. MD2FILE publishes the series and provides the editor mentioned here; that commercial interest does not establish the suitability of a format for every workflow.

What can travel inside a Markdown file?

A Markdown file can hold prose and structural conventions that remain understandable in a text editor. CommonMark provides a specification for interpreting that syntax. Its account of historical parser differences also explains why a document can render differently across implementations. A declared parser or dialect makes the expected behavior clearer. CommonMark specification.

Some familiar features belong to extensions. GitHub Flavored Markdown adds tables and task lists, among other features, and GitHub applies further processing to the resulting HTML. A `.md` suffix therefore gives an incomplete description of a document's rendering requirements. GFM specification.



Conceptual path from Markdown source through dialect, assets and rendering to a shareable document.

This conceptual diagram separates the text file from the conditions needed to display it. Source can survive a move even when a diagram renderer, linked image or application-specific feature is missing.

Part of a note	What the file can preserve	What needs a separate check
Prose and headings	Wording and visible structural markers	Heading hierarchy and generated anchor links
Code examples	The characters inside a fenced block	Language highlighting and line wrapping
Images	A path or URL, plus alternative text	The image file, access permissions and resolution
Tables and tasks	Source rows and task markers	Support in the destination dialect
Diagrams and equations	Source notation	Compatible rendering software and fonts
Application features	Sometimes a reference or special marker	Queries, embeds, plugin behavior and local configuration

The table is a review aid. It does not imply that every application loses the same features or that a successful import preserves every meaning.

File ownership and application independence

Obsidian documents a concrete file-based model: notes are Markdown files in a local vault folder, and other editors can modify them. The application also maintains separate settings and a metadata cache. This illustrates why retaining readable files and retaining the complete application experience are different tasks. How Obsidian stores data.

For a long-lived knowledge collection, record which parts are ordinary files and which parts depend on software behavior. A folder containing notes and images is easier to inspect than a folder of notes that silently depends on attachments elsewhere. Keep a short description of required extensions beside the collection, particularly if a colleague will maintain it after the original author leaves.

Plain text also exposes changes to review. A reviewer can inspect a changed sentence without interpreting a page-layout file. That does not make every change meaningful: automatic wrapping or regenerated output can still obscure substantive edits. Teams should agree on a small set of formatting conventions that make their own reviews manageable.

The existing Markdown documentation guide covers the underlying writing patterns. Portability adds a further concern: whether those patterns carry enough context beyond the original workspace.

A README is a document inside a repository

GitHub resolves relative links and image paths using the file's location and branch. It also creates navigation from headings. When the README leaves that environment, a converter or publishing process must establish its own base location for relative references. GitHub's README documentation.

Consider an illustrative handbook package with three files:

```
handbook/
  README.md
  docs/
    setup.md
  images/
    architecture.svg
```

The README contains these references:

```
# Project handbook

Read the [setup instructions](docs/setup.md).

![The project's components and their connections](images/architecture.svg)
```

`docs/setup.md` contains the setup instructions, and `images/architecture.svg` contains the diagram. Neither file is embedded in the README by those references. The package needs all three files; a `.md` file by itself carries only the link destinations and image alternative text.

Suppose the sender copies only `README.md` into `delivery/README.md`. In a viewer that resolves paths relative to that file, `docs/setup.md` now points to `delivery/docs/setup.md`, and the image points to `delivery/images/architecture.svg`. Those destinations do not exist in the one-file delivery. The original files remaining under `handbook/` cannot satisfy those paths for a recipient who never received that folder.

For an editable offline handoff, the resolution is to copy the two dependencies with their relative paths intact:

```
delivery/
  README.md
  docs/
    setup.md
  images/
    architecture.svg
```

The README links can now stay unchanged. The file inventory is complete for this example because the README has exactly these two dependencies, and we assume the setup page and SVG have no further external references. In a real package, follow references inside those files too. Open `delivery/README.md` in the intended viewer with `delivery/` as its directory context, follow the setup link and inspect the diagram. A converter that accepts only pasted text needs its own asset-loading arrangement; this folder structure alone cannot give it access to local files.

This is a worked path-resolution example, not a recorded application test. A recipient who receives only a PDF needs a different handoff: include the essential setup instructions in the report or replace the relative link with a maintained destination they can access. Check the actual PDF's image and link behavior separately. Review access permissions before sharing material outside the team.

Our GitHub-Flavored Markdown export guide includes a downloadable README example. It is a practical way to inspect how source, an image and the resulting PDF relate; it is not evidence that every repository document exports identically.

Preserve the decision behind the prose

A note often relies on knowledge shared by its original readers. “Use the second option” may be clear beside a live conversation and meaningless six months later. Portable knowledge needs enough context to be understood without that surrounding screen.

For example, an illustrative engineering decision record could include:

```
## Decision: keep the existing import format

Status: accepted for the next release
Reason: the proposed replacement omits attachment references.
Scope: documentation imports; other integrations are unchanged.
Evidence: link to the reviewed sample and issue.
Revisit when: the replacement preserves the required references.
```

The value comes from the explanation and references, not the filename extension. Add the actual decision date and responsible owner when creating a real record. Separate a decision from a suggestion, and identify what evidence could change it. Avoid leaving example placeholders in a published document.

For knowledge that also directs an AI tool, instruction scope and authority introduce additional concerns. Those belong in Markdown and AI agent instructions, rather than being assumed from the readability of a note.

Choose the output for the next reader

Reader's task	Useful handoff	Main review question
Continue editing	Markdown with required assets	Can the recipient edit and rebuild it?
Read a maintained reference	HTML with stable navigation	Are links, headings and revisions understandable?

Reader's task	Useful handoff	Main review question
Review a fixed issue of a report	PDF plus its source where appropriate	Are pages legible, searchable and clearly versioned?
Reuse measurements	CSV or JSON with definitions	Are units, missing values and populations explicit?

Several outputs can coexist. A report can have a canonical HTML page, a dated PDF and a small data download. Keep their version information consistent so a correction in one format does not leave the others looking authoritative but stale.

The MD2FILE editor is one option for reviewing Markdown before producing PDF or HTML. Standard document processing happens in the browser, while optional AI and cloud features have different data paths; consult the privacy policy for that boundary. The PDF review checklist covers the output checks that remain after the text is ready.

Before handing over a knowledge package, open it outside the original workspace, follow its essential links and inspect its images. Ask a reader who lacks the original context to identify the decision, evidence and current status. Record the format, assets and version that passed that review so the next revision has a concrete starting point.

CHAPTER 6

How to Check a Markdown PDF Before Sharing

Before sharing a PDF made from Markdown, open the downloaded file and check that its images are present, diagrams are readable and page breaks preserve the reading order. Search for a sentence and copy a code sample to check whether the text remains usable. A successful export alone does not answer those questions.

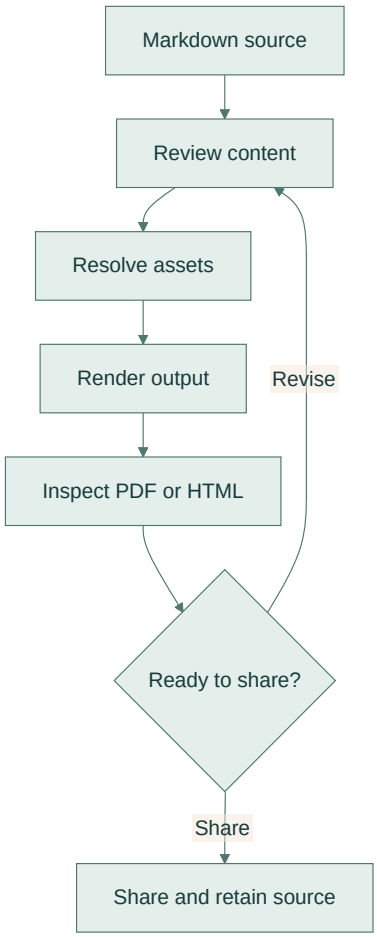
This chapter is a review checklist for an output you have already generated. The conversion methods guide covers choosing a tool and exporting the source. Here the task is to decide whether the resulting PDF is ready for its reader. The recommendations draw on format and platform documentation; they are not results from a comparative converter test.

It is part of The Markdown Layer, published by MD2FILE. The editor and extension described below are our products. Their inclusion illustrates workflow choices, not an independent ranking or a promise of identical output across converters.

Rendering adds decisions that Markdown leaves open

The conversion path determines how source becomes pages. Pandoc, for example, documents a default LaTeX route to PDF and alternative intermediate formats, including HTML. Its styling options depend on the selected path. This is one reason the same Markdown source can produce different documents in different tools. Pandoc's PDF documentation.

Paged media introduces its own controls: page dimensions, margins, breaks and running material. Screen layout does not settle those choices. A wide code block may be comfortable in a horizontally scrolling preview and unsuitable for a narrow printed page. CSS paged media documentation.



Conceptual workflow from source review through rendering and pagination to output inspection, sharing and source retention.

The workflow is a proposed review process, not a measured funnel. Inspection follows pagination because a heading, table or caption can move when the document gains page boundaries.

Start with assets and explicit requirements

Write down what must survive before choosing an export mode. For a technical report, that might mean selectable code, readable equations, an intact table and diagrams whose labels remain legible at the intended page size. A visual match to a webpage and editable text may require different compromises.

Element	Inspect before export	Inspect in the downloaded file
Images	File location, loading and alternative text	Presence, resolution and relationship to captions
Mermaid diagrams	Successful rendering and readable labels	Clipping, scaling and page placement
Equations	Correct notation and supported rendering	Missing symbols, baseline and line breaks
Tables	Clear headers and manageable width	Split rows, repeated context and legible text
Code	Correct characters and language labeling	Wrapping, truncation and copied text
Links	Correct destination and access	Clickable destination where supported

Remote images deserve a separate check. Access rules or browser cross-origin restrictions can prevent an image from being fetched during conversion. A preview that displayed an image earlier is insufficient proof that an exporter can retrieve it. Prefer an asset package or deliberately accessible images when the workflow supports them, and inspect the resulting file.

The README-to-PDF guide has a small source-and-PDF example. For diagram notation and formula checks, use the existing Mermaid and LaTeX guide. Documents with Chinese, Japanese or Korean text also need a font coverage review.

Review page breaks with an actual reading task

An illustrative failure is a “Workflow” heading at the bottom of a page with its diagram on the next page. Both elements exist, but their separation makes the document harder to follow. A reader should not need to guess whether the following figure belongs to that heading or the next section.

Possible repairs include reducing an oversized figure, shortening its surrounding text or adjusting the content order. Select the smallest change that preserves meaning. Avoid shrinking an entire report until the labels become unreadable just to achieve a preferred page count.

For a long report, inspect the first page, every section transition and pages containing wide or tall material. Also search for a sentence near the end and copy a representative code sample from the PDF. These checks answer different questions: an attractive screenshot does not prove that the text can be retrieved correctly.

A visually readable PDF is not automatically an accessible or archival-standard PDF. W3C guidance for complex images calls for both a short identification and a fuller text equivalent. Keep the chart's meaning and essential data available as text alongside the visual. W3C complex-image guidance.

Editing and existing-file conversion are different jobs

When the source needs changes, the MD2FILE web editor provides an editing and preview step before PDF or HTML export. Save current work before importing another document. The Markdown-to-PDF methods guide explains the existing controls and shows a downloadable example.

When a `.md` file is already ready, the MD2FILE Markdown extension provides a separate file-to-PDF workflow. Its public listing describes browser processing with Mermaid, KaTeX and optional CJK support. These capabilities still require review on the actual document; “supports diagrams” does not guarantee that every diagram fits every page size. Chrome Web Store listing.

Browser-side generation also has a narrower meaning than “no network activity.” MD2FILE's standard web PDF path builds the file locally, while usage metadata and optional account, AI or cloud features have separate handling. Remote assets can require requests. The web privacy policy explains the document-content boundary. Extension privacy and permissions should be assessed for the installed version rather than inferred from the website's policy.

If automation, repeated builds or citation processing is the main requirement, a command-line publishing workflow may be more suitable. The existing Pandoc comparison discusses that decision. Saved HTML introduces further layout and resource questions, covered in the HTML-to-PDF guide.

Keep a reproducible handoff

Retain the source, required assets and the version of the released output together. Record the conversion path and any settings that materially affect the document. When an output contains an attached copy of its source, inspect that attachment too; removing visible text from a later presentation is not a reliable way to remove information from the original source.

For a research report, keep a small release note identifying its date, version and corrections. Give readers a canonical web page for updates and a dated download for the fixed edition they used. That arrangement lets a citation refer to a particular document without concealing later corrections.

CHAPTER 7

AI Conversations as Documents: Preserve Context, Sources and Decisions

An AI conversation can contain material worth keeping: an explanation, a draft, a comparison or a proposed decision. Once that material leaves the chat, the reader may lose the prompt, corrections and source context that made it understandable. A useful export therefore needs editorial review as well as a download button.

Vendors now provide several ways to move generated material into documents. That establishes the availability of the workflow. It does not tell us how frequently people use it, whether the exported claims are correct, or whether a full transcript is the most useful record.

This chapter of The Markdown Layer examines those choices. MD2FILE publishes the series and offers the AI Chat Exporter discussed below. The review examples are illustrative advice, not results from a user study or a comparative product benchmark.

Native exports already connect chat and documents

Google documents exporting a Gemini response to a new Google Doc, and exporting eligible tables to Sheets. Availability varies with the app and account settings, and the destination service's terms apply to the exported material. These options show a direct route from an answer to a document or dataset that can be edited elsewhere. Gemini export documentation.

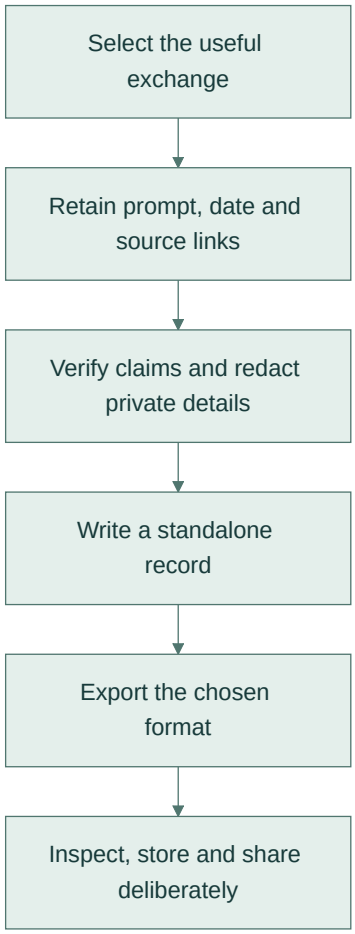
Anthropic's current artifact documentation describes document exports to Word, PDF, Markdown and Google Docs, alongside download or copy controls for legacy artifacts. An artifact can stand apart from the surrounding conversation, which makes its retained context worth checking before sharing. Claude artifact documentation.

Native document exports and transcript exports serve different purposes. A generated report may be intended as the finished deliverable; a transcript records an exchange that led to it. Choose according to what the recipient needs to inspect. A reviewer assessing a conclusion may need selected prompts and corrections even when the final document reads well on its own.

Decide which part of the conversation belongs in the record

Material to preserve	A useful output	Context to retain
A reusable explanation	Edited note or selected response	Question, assumptions and intended audience
A decision discussion	Decision record with supporting excerpts	Alternatives, unresolved objections and approver
Research assistance	Report with checked references	Source URLs, dates and evidence limitations
A troubleshooting exchange	Relevant transcript range	Environment, attempted fixes and final observed result
A draft for continued work	Editable document or Markdown	Draft status, outstanding checks and source assets

A long transcript can preserve detail while making the conclusion difficult to find. A short extract can be convenient while hiding a correction. If a response was revised after the user supplied new evidence, retain that correction or state what changed. Mark omitted material when its absence could affect interpretation.



Conceptual workflow for selecting a chat exchange, retaining context, verifying and redacting it, then exporting and reviewing a standalone record.

This is a proposed workflow. Verification and redaction happen before export, followed by a check of the file that will actually be sent. The diagram does not imply that exporting a conversation performs either review automatically.

Preserve provenance without copying everything

A standalone record should identify its purpose and the circumstances relevant to the content. Depending on the task, that may include the conversation date, the assistant used, the exact question and the source material supplied. Record model or tool details only when they are known and useful; do not guess a model version from its writing style.

Consider this fictional exchange about a team handbook. It illustrates an editorial correction, not an actual user conversation or a compatibility test:

> **User:** Can we put a table in the Markdown handbook and expect every CommonMark viewer to display it?
>
> **Assistant, first answer:** Yes. Tables are part of CommonMark.
>
> **User:** The GFM specification calls tables an extension. Have we checked whether our viewer supports that extension?
>
> **Assistant, correction:** My first answer was too broad. GFM defines a table extension. We have not checked the destination viewer, so its support is still unknown.

The correction is supported by the GitHub Flavored Markdown specification, which defines tables as an extension. Exporting only the first answer would preserve the error. A useful edited record keeps both the corrected claim and the unresolved check:

```
## Handbook table format

Status: proposed; destination viewer compatibility is untested.
Proposed approach: use a GFM table if the handbook's viewer supports it.
Correction retained: the initial answer incorrectly treated tables as
part of core CommonMark. GFM defines them as an extension.
Source: https://github.github.com/gfm/
Next check: open a sample in the intended viewer and inspect its
headers and cell contents before approving this format.
```

This record does not claim that the check passed or that the team accepted the proposal. A real record should also retain the conversation date and responsible reviewer. If the deliverable is a transcript, include the correction turn with the earlier answer; if it is an edited brief, state the correction as the example does. Link any actual test report separately from the assistant's account of it.

The same discipline applies to research citations. Open the source and check that it supports the particular claim, including its date and population. Keep the source URL beside the claim in the exported document. A bibliography at the end helps readers find sources, but it does not resolve an ambiguous claim halfway through a long transcript.

For a broader treatment of sources that move between applications, see Markdown knowledge portability. Instructions that direct future agent behavior require a separate review of scope and conflicts, covered in maintaining agent documentation.

Review privacy at each stage

Chat content may include information that belongs in a working session but not in a shared record. Review prompts as well as answers. Names, private links and pasted credentials can appear in a question, a code block or a quoted source even when the final explanation looks harmless.

Hiding prompts is a presentation choice. It does not guarantee that an answer has stopped repeating details from those prompts. Search the finished document for the information you intended to remove, and inspect any included attachments or source files before sharing it.

Local PDF generation describes where the file is assembled. It does not undo the chat provider's prior processing, determine where a cloud destination stores the file or establish that an extension makes no network requests. Chrome's permission documentation explains that host permissions can enable page access and extension requests. Assess permissions and the relevant privacy policy for the installed product. Chrome extension permissions.

A browser export option for an existing conversation

The MD2FILE AI Chat Exporter has Chrome and Safari editions for producing PDFs from supported conversations. Both offer a free export allowance and controls for selecting one response, a recent range or the full conversation. Their supported sites and account services differ.

The Chrome edition advertises support for ChatGPT, Claude, Gemini, Grok, DeepSeek, Copilot and Perplexity. Its Chrome Web Store listing describes PDF output and controls for including prompts. AI Chat Exporter for Safari on Mac supports ChatGPT, Claude and Gemini. These are the advertised capabilities checked on October 15, 2026; compatibility can change when a chat service changes its interface. We have not tested every current browser/provider combination. Neither edition promises Markdown or Word output.

The Chrome privacy policy describes hosted billing, account linking and subscription metadata, and states that exported chat content is not stored on its billing server. The Safari privacy policy describes PDF processing on the Mac and purchases handled by Apple, without a separate Safari account login or external billing service. It also

explains that including chat images may require requests to their source hosts. Local PDF generation does not mean the entire workflow makes no network requests.

For material that needs substantial rewriting, an editable source may be preferable before the final PDF. A Markdown note can be reviewed in the MD2FILE editor or another editor, with its cited sources and necessary assets retained. The PDF review checklist covers the later checks for figures, tables and pagination.

Check the shared file against the intended record

Open the downloaded file separately from the chat. Confirm that the selected range starts and ends where intended, corrections are present, and links still point to the sources that support the claims. Inspect code characters and table columns; formatting can make an otherwise intact passage difficult to use.

Then ask whether a recipient can distinguish a verified observation, an assistant suggestion and an accepted decision. Label unresolved questions directly. Store the reviewed file with a date and version, and retain the editable source when future correction is likely. If the document changes, issue an updated version rather than leaving readers to infer which copy is current.

CHAPTER 8

The Markdown Layer: Methodology and Research Data

The Markdown Layer combines published research, platform documentation and a new, deliberately narrow GitHub repository snapshot. These sources answer different questions. We keep their populations and dates separate, identify missing observations, and distinguish measured file presence from interpretations about how people or agents use Markdown.

This methodology accompanies the research hub and the repository study. The original version 1.0 is dated September 30, 2026. MD2FILE publishes the report and offers document-conversion products; the research is not an independent product certification, user survey or market-size estimate.

Evidence classes and source review

Published studies retain their authors' populations, dates and measurement definitions. An eligible-repository sample does not automatically describe all GitHub projects. A website listing, a developer announcement and an experiment with an AI agent also measure different things. We use vendor documentation to describe documented behavior and research papers to describe their reported observations or experiments.

The source inventory identifies references used across the series. For a quantitative claim, the relevant record is the original paper, dataset or platform publication wherever available. Access to a paper does not imply that we independently reran its analysis. Where a source dataset could not be retrieved, the distinction remains explicit.

For example, Hora, Montandon and Costa's 2026 repository-content paper uses a random sample within an eligibility-filtered population. Its historical observations are subsets of repositories surviving into its 2026 sample. Our new panel uses current star ordering and one pinned commit per repository. The two designs cannot be pooled into a trend line or a shared adoption percentage.

Conceptual diagrams in the report explain workflows. They are not measured flows or estimates of how many people move through each stage. Product references are examples whose claims require separate verification; they do not contribute observations to the repository study.

Original panel selection

We selected six repositories from each of four GitHub language-filtered searches. The exact query template was:

```
language:{language} stars:>=1000 pushed:>=2026-01-01 fork:false archived:false is:public
```

`{language}` took the values `Python`, `TypeScript`, `Rust` and `Go`. The REST search parameters were `sort=stars`, `order=desc`, `per_page=6` and `page=1`. We retained GitHub's returned order, including any tie order, and required `incomplete_results=false`. All 24 selected repository identifiers were distinct. See GitHub's repository-search definitions.

These are named, highly starred repositories satisfying the search criteria on September 30, 2026. The sample includes curated lists, educational material and agent-related projects. Its membership is therefore unsuitable for estimating prevalence among ordinary projects, all developers or private repositories. Language filters are not demographic categories.

The search and file-tree collection ran from **13:52:29 to 13:54:49 UTC**. For each selected repository, we resolved the current default branch to its latest commit once, retained that commit's tree SHA and requested the recursive tree by that SHA. Stars, language and other repository metadata belong to the search observation; tree contents belong to the pinned commit. They are not a historical time series.

API budget and incomplete observations

The original collection made **53 read-only GET requests**: one rate-limit preflight, four repository searches, 24 commit lookups and 24 tree requests. It used GitHub REST API version `2022-11-28`, optional existing authentication, a 40-second request timeout and no automatic retries. Tokens were held in memory and were not written to the dataset or logs.

The collector limited each response to **12 MiB**, reading one extra byte to detect an oversized body. Three tree responses exceeded that limit: `openclaw/openclaw`, `rust-lang/rust` and `microsoft/TypeScript`. No complete tree was parsed for these repositories. They remain in the selected panel with unknown file counts; the other 21 trees were complete.

The first oversized response stopped the initial collector before its received prefix was saved. We retained the stopped-run record and recorded that request without inventing an HTTP status, full response size or body hash. The bounded continuation reused saved results and collected only untouched repositories. It did not repeat the oversized request or select a replacement.

GitHub also documents its own recursive-tree limits and `truncated` flag. A client byte ceiling and GitHub's truncation flag are different checks. Here the three unknown trees were stopped by the client ceiling. We do not infer whether their complete server responses would have carried a truncation flag.

File definitions

Counts use regular tracked Git blobs with modes `100644` or `100755`. Symlink targets and submodule contents are outside the count. Git LFS pointer size, where present, describes the tracked pointer rather than the linked object's full size.

Field	Definition
Markdown files	Paths ending `.md` or `.markdown`, case-insensitive
Markdown bytes	Sum of those blobs' reported sizes
MDX files	`.mdx` paths, counted separately

Field	Definition
Root Markdown README	Root `README.md` or `README.markdown`, case-insensitive
Human-document names	README, CONTRIBUTING, CHANGELOG, CODE_OF_CONDUCT or SECURITY with either Markdown extension, anywhere, case-insensitive
Root docs directory	Exact tree path `docs`
Named context candidate	Exact `AGENTS.md`, `CLAUDE.md` or `GEMINI.md` basename anywhere, or `.github/copilot-instructions.md`
Nested context candidate	A recognized candidate with at least one slash in its path

The dataset also records `SKILL.md`, `.github/instructions/*.instructions.md` and `.cursor/rules/*.mdc` separately. These names are not added to the headline context-candidate count. The file-path CSV flags several directory names that may indicate tests, examples or vendored material. That flag is a reading aid, not a validated classification of whether a file is active.

A repository can contain several categories. Counting `AGENTS.md` and `CLAUDE.md` repositories separately and adding the results would double-count repositories containing both. The dataset includes exact category combinations so that charts can show overlap honestly.

Denominators and missing data

The selected panel has **24 repositories**, of which **21 have complete trees**. Aggregate file counts and category-presence counts use those 21. JSON `null` and blank CSV fields mean unknown; a recorded zero in a complete tree means the exact matching rule found no such file.

The 35,904 Markdown files and 152,448,261 source bytes are totals across complete trees. Larger documentation repositories dominate those totals. The study does not estimate prose quality, readership, active use, agent effectiveness, conversion demand or historical growth. No third-party Markdown was rendered as part of this panel.

Downloads and reproduction

The public artifacts contain repository metadata, identifiers, paths and derived counts. They omit third-party file contents, commit-author emails, credentials and private operating notes.

- Repository dataset and definitions, JSON
- One row per selected repository, CSV
- Classified filenames with immutable source links, CSV
- Python reproduction script

The script uses Python 3.10 or newer and its standard library. Its default mode checks the published metadata without making a network request:

```
python3 reproduce-panel.py --manifest repository-panel.json
```

To recompute the original complete observations from immutable GitHub trees, use an empty output directory and explicitly enable the network mode:

```
python3 reproduce-panel.py --manifest repository-panel.json \
  --fetch-pinned --output reproduced-panel
```

This mode makes at most **22 GET requests**: one rate check and 21 pinned-tree requests. It preserves the three original unknowns and does not retry failures. Optional credentials can come from `GH_TOKEN`, `GITHUB_TOKEN` or an existing GitHub CLI login. A separate offline mode accepts privately saved raw tree responses.

We ran that offline tree-recomputation mode against the original saved responses. All 21 complete trees matched the published derived fields. This verifies the file-count calculation, not the correctness of repository content. Repeating today's search later may select different projects; use the pinned identifiers to reproduce this observation.

Citation, reuse and corrections

Cite the publisher, report title, version 1.0.1 dated October 15, 2026 and the relevant study or methodology URL. This editorial revision clarifies the Chrome and Safari exporter editions; it does not change the September 30 repository panel, calculations or research findings. BibTeX and plain-text citation downloads accompany the full PDF and report source.

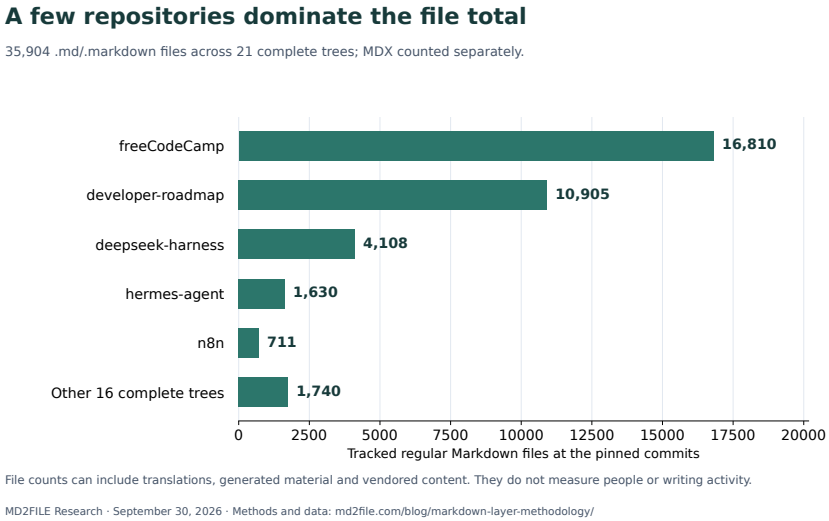
Repository license metadata is retained as provenance. The download does not relicense underlying repositories, and this release makes no separate license grant for the dataset or scripts. Check the relevant terms before redistributing content or code.

A correction should identify the dataset version, repository, commit and disputed field. Future revisions should record changed definitions or coverage rather than silently replacing an unknown with zero. Any 2027 outlook in the series is editorial interpretation, not a measurement collected in 2027.

CHAPTER 9

File concentration and the 2027 research agenda

File concentration



Markdown file concentration in the panel: freeCodeCamp 16,810; developer-roadmap 10,905; deepseek-harness 4,108; hermes-agent 1,630; n8n 711; the other 16 complete trees 1,740.

Repository or group	Regular `.md` / `.markdown` files
freeCodeCamp/freeCodeCamp	16,810
nilbuild/developer-roadmap	10,905
deepseek-ai/deepseek-harness	4,108
NousResearch/hermes-agent	1,630
n8n-io/n8n	711
Other 16 complete trees	1,740
Total across 21 complete trees	**35,904**

A translation, fixture, generated page or vendored document can contribute to that total. We did not read every file, classify authorship, or measure monthly writing activity. The distribution is evidence of concentration within this panel. It cannot tell us how many people adopted Markdown or how much AI increased document creation.

Questions for the next edition

The evidence does not justify declaring a winner among instruction formats or predicting export demand. A useful next edition would test narrower questions against this baseline.

Question	Evidence needed	What would change our interpretation?
Do instruction files remain actively maintained?	Repeated observations at pinned commits, with change review	Files persist but their guidance becomes stale
Does shared guidance reduce duplication?	Content-level analysis with explicit permission and scope rules	Multiple filenames contain conflicting instructions
Does portable source survive real transfers?	Repeatable fixtures across named tool versions	Text survives while required assets or semantics fail
Which conversations become documents?	Consented workflow research with a defined population	Export use is narrow or mainly archival
Does an instruction change help a team?	Local task outcomes, effort and review quality	Added context increases work without useful gains

These are research proposals. No 2027 observations are included in this edition, and the current panel should not be silently expanded and compared as though it were a fixed cohort. A future update should state which repositories were retained, replaced or still unobservable and why.

CHAPTER 10

Source register and editorial disclosure

Editorial disclosure: MD2FILE publishes this report and sells tools discussed in it. AI assisted research, drafting, code and the cover illustration; the source checks, retained data and limitations are documented so readers can inspect the claims. Charts are calculated from the published panel. Workflow diagrams are conceptual, created with Mermaid, and do not represent measured traffic or user journeys. Product documentation and source inspection are not a security audit or a test of every current browser/provider combination. This edition includes no user survey, private usage analysis or causal SEO result.

The dated machine-readable source register records study populations, source types and reproduction limits. Different versions of a study are not independent evidence. Original repository contents are not republished in this report.

- **A1: What's Inside a GitHub Repository? An Empirical Study on the Contents of 10K Projects.** Andre Hora, João Eduardo Montandon, Diego Elias Costa 2026-07-13. Source. Reviewed September 30, 2026.
- **A2: What 7,370 popular GitHub repositories put in AGENTS.md and CLAUDE.md.** Stride Research 2026-09-27. Source. Reviewed September 30, 2026.
- **A3: Agentic Much? Adoption of Coding Agents on GitHub.** Romain Robbes, Théo Matricon, Thomas Degueule, Andre Hora, Stefano Zacchiroli 2026-04-08. Source. Reviewed September 30, 2026.
- **A4: Configuring Agentic AI Coding Tools: An Exploratory Study.** Matthias Galster, Seyedmoein Mohsenimofidi, Jai Lal Lulla, Muhammad Auwal Abubakar, Christoph Treude, Sebastian Baltes . Source. Reviewed September 30, 2026.
- **A5: On the Impact of AGENTS.md Files on the Efficiency of AI Coding Agents.** Jai Lal Lulla, Seyedmoein Mohsenimofidi, Matthias Galster, Jie M. Zhang, Sebastian Baltes, Christoph Treude 2026-03-30. Source. Reviewed September 30, 2026.
- **A6: Evaluating AGENTS.md: Are Repository-Level Context Files Helpful for Coding Agents?.** Thibaud Gloaguen, Niels Mündler-Sasahara, Mark Niklas Müller, Veselin Raychev, Martin Vechev 2026-09-29. Source. Reviewed September 30, 2026.
- **A7: Markdown Mayhem: Taming the Agentic Documentation Explosion.** Harsha Kokel . Source. Reviewed September 30, 2026.
- **A8: Current instruction format and loading documentation.** AGENTS.md project, Anthropic . Source. Reviewed September 30, 2026.

- **W01: Chrome I/O 2026 recap.** 2026-05-22. Source. Reviewed September 30, 2026.
- **W02: Chrome-Stats category benchmarks.** 2026-07-31. Source. Reviewed September 30, 2026.
- **W03: Chrome-Stats methodology.** . Source. Reviewed September 30, 2026.
- **W04: CommonMark 0.31.2 specification.** 2024-01-28. Source. Reviewed September 30, 2026.
- **W05: GitHub Flavored Markdown specification.** 2019-04-06. Source. Reviewed September 30, 2026.
- **W06: About repository READMEs.** . Source. Reviewed September 30, 2026.
- **W07: How Obsidian stores data.** . Source. Reviewed September 30, 2026.
- **W08: Pandoc: creating a PDF.** . Source. Reviewed September 30, 2026.
- **W09: CSS paged media.** . Source. Reviewed September 30, 2026.
- **W10: W3C complex images.** . Source. Reviewed September 30, 2026.
- **W11: Export responses from Gemini Apps.** . Source. Reviewed September 30, 2026.
- **W12: Claude artifacts.** . Source. Reviewed September 30, 2026.
- **W13: Chrome extension permissions.** . Source. Reviewed September 30, 2026.
- **W14: MD2FILE Markdown extension listing.** 2026-04-01. Source. Reviewed September 30, 2026.
- **W15: MD2FILE AI Chat Exporter listing.** 2026-05-18. Source. Reviewed October 15, 2026.
- **W16: AI Chat Exporter privacy policy.** 2026-05-04. Source. Reviewed October 15, 2026.
- **W17: MD2FILE web privacy policy.** . Source. Reviewed September 30, 2026.
- **M01: Markdown.** John Gruber . Source. Reviewed September 30, 2026.
- **D01: MD2FILE September 2026 repository panel.** 2026-09-30. Source. Reviewed September 30, 2026.
- **G01: Git trees — GitHub REST API.** . Source. Reviewed September 30, 2026.
- **G02: Searching for repositories.** . Source. Reviewed September 30, 2026.
- **W18: AI Chat Exporter for Safari App Store listing.** . Source. Reviewed October 15, 2026.
- **W19: AI Chat Exporter for Safari privacy policy.** 2026-08-24. Source. Reviewed October 15, 2026.

Citation

MD2FILE Team. (2026). *The Markdown Layer: State of Markdown 2026*. Version 1.0.1, October 15. https://md2file.com/blog/the-markdown-layer/