ATLORIA

Using Code Parsing Results to Support Technical Docs

August 25, 2026

Identifying Which Parsing Outputs Belong in Your Docs Workflow

After you review coverage in Reviewing Parsed Code Results and Reference Coverage, the next job in Atloria is deciding which parsed results should feed your documentation work. In the Code Parsing workspace, focus on results that can directly support pages your team already maintains: file paths, exported items, route entries, inline descriptions, type details, imports, and configuration keys. These are the most useful starting points because they describe what exists in the codebase without requiring writers to inspect every file manually.

A practical way to organize this in Atloria is to map each result type to a documentation outcome:

Parsing result	Best documentation use
File paths and folder structure	Project structure pages and onboarding docs
Exported items and signatures	Technical reference pages
Route entries	Endpoint and navigation-related docs
Imports and dependencies	Architecture and relationship pages
Inline descriptions	Draft summaries for reference content
Configuration keys	Setup and configuration docs

Keep a clear line between parser facts and writer judgment. Parser facts are items Atloria can surface directly from uploaded code, such as a file location, an exported name, a route path, or a parameter list. Writer judgment is everything that explains meaning: when to use something, why it matters, what readers should avoid, and how it fits a workflow. Use parsed results to build the skeleton of a page, but treat explanations and recommendations as editorial work.

Before your team uses parsed results in a draft, make sure each item includes enough identifying detail to be trustworthy. At minimum, confirm the result shows the

repository path, language, item name, signature or visible structure, visibility level if available, and source location. If any of those details are missing in Atloria's parsing results, the item may still help with research, but it is not a strong candidate for direct documentation drafting.

[SCREENSHOT: Code Parsing workspace showing parsed result types such as file paths, exports, routes, and configuration entries]

Turning Parsed Code Data into Technical Reference Pages

In Atloria, parsed results are most valuable when you turn them into structured reference pages instead of leaving them as raw findings in the Code Parsing workspace. Start with exported items and signatures. These results usually give you the page title, where the item lives, what inputs it expects, what it returns, and how it is exposed to other parts of the project. That information can fill the core sections of a reference page: name, source file, inputs, outputs, visibility, and related items.

When Atloria shows grouped exports by file or package area, use that view to plan parent-and-child reference pages. A higher-level page can describe a folder, package, or module area, while linked child pages cover individual exported items. If the parsed results show relationships such as inheritance, implemented interfaces, or shared types, include those as "Related reference pages" links so readers can move naturally between connected entries instead of searching from scratch.

Inline descriptions are helpful, but they should be treated as draft material. If Atloria surfaces comments or short descriptions from the code, pull them into a draft summary and then review the wording before publishing. Some descriptions are written for developers and may be too brief, too internal, or too inconsistent for end-user technical docs. If an exported item has no inline description at all, flag it for writer follow-up rather than publishing a nearly empty page.

You can also use relationship data to improve navigation. Imports, shared types, and call links often reveal which reference pages belong together. In Atloria, connect these pages with links such as "Used by," "Depends on," or "Related pages" where that relationship is clearly supported by the parse results. This makes your reference section easier to browse and reduces duplicate explanations across pages.

[SCREENSHOT: Draft technical reference page in Atloria populated from parsed exports and linked related entries]

Using Parse Results to Explain Project Structure and System Behavior

Parsed results in Atloria are not only for reference pages. They also help you explain how a project is organized and how work moves through it. Instead of manually listing folders and guessing responsibilities, use the parsed directory tree, entry files, and grouped file areas to build project structure pages. These pages are especially useful for onboarding writers, reviewers, and subject-matter experts who need a quick map of the repository before editing documentation.

Start with the broadest visible structure: top-level folders, major feature areas, and known entry points. In Atloria, these parsed results can help you describe where account access screens live, where admin workspace pages appear, where project pages are grouped, and where public documentation views are defined. That gives readers a practical orientation without requiring them to inspect code line by line.

Dependency and import views are useful for explaining interaction between areas of the project. If one area consistently depends on another, document that relationship as part of the project structure story. If Atloria reveals unexpected cross-connections, note them as review points for your team. These findings can help you explain why a workflow touches several areas, such as account access, project setup, document editing, analytics, and publishing.

Route-related parse results also help you describe user-facing flows. For example, if Atloria shows routes for login, registration, admin analytics, project analytics, or webhook-related pages, you can use those findings to map visible navigation paths and screen groupings. Configuration keys are another important source. When parse results reveal named settings that affect workflows, use them to identify setup topics your docs should cover, especially when behavior changes by project, organization, or connected feature.

The goal is not to reproduce a code map. It is to turn parsed structure into readable documentation that explains how Atloria’s screens, workflows, and connected areas fit together.

Planning Documentation Coverage from Gaps in the Parsed Data

One of the best uses of parsed results in Atloria is finding what your documentation does **not** cover yet. Gaps become easier to spot when you compare parsed outputs with your existing docs set. Publicly exposed items with no description, route entries

with no matching page, and configuration keys with no setup guidance are all strong signals that your team has missing documentation work.

Use the Code Parsing workspace to sort findings by likely impact. Start with items that appear central to the reader experience: visible routes, heavily connected exports, shared types, and files near entry points. These usually affect more pages and more workflows than isolated items buried in a narrow feature area. If Atloria shows import frequency, dependency relationships, or placement in core project areas, use those signals to rank what should be documented first.

Grouping also matters. Rather than assigning work file by file, create documentation sets based on visible project areas in the parsed results. For example, you might group account access pages together, admin workspace pages together, project analytics pages together, and integration-related pages together. This makes assignments cleaner and helps writers stay consistent within a topic area.

Parsed results are also useful for drift tracking. When a file moves, a visible item is renamed, a route changes, or a parameter list no longer matches your published page, your documentation may already be out of date. Comparing current parse results with published content in Atloria helps you catch those changes before readers do. This is especially important for technical reference pages, setup instructions, and workflow documentation that depends on exact names and locations.

If your team already publishes reference content, use parsed gaps as a planning queue rather than a one-time cleanup list. That turns code parsing into an ongoing documentation maintenance process.

[SCREENSHOT: Atloria documentation planning view with parsed gaps grouped by feature area and priority]

Reviewing and Enriching Parser-Derived Drafts Before Publishing

Parser-derived drafts in Atloria can save time, but they should never be treated as publish-ready without review. Before a draft becomes part of your project documentation, verify that the parsed details still match the current source. Check the source location, visible signature, and the branch or repository state used for the parsing run. If the parse results came from an older upload or a previous workspace session, the draft may already be stale.

Once the factual details are confirmed, add the context that parsing cannot reliably provide. Atloria can help surface names, relationships, and structure, but it cannot fully

explain business meaning, audience expectations, workflow consequences, or when a reader should choose one option over another. Writers still need to add plain-language explanations, usage notes, limitations, and examples that fit the project's audience.

This review step is also where you standardize naming. Parsed results often reflect inconsistencies in the source material, such as mixed naming styles, uneven descriptions, or overlapping patterns across files. In Atloria, keep your published page titles, section names, and cross-links consistent even if the parsed source data is not. That makes the documentation easier to scan and reduces confusion for readers moving between related pages.

A simple editorial workflow helps keep quality high:

1. Confirm the parsed facts match the current code results in Atloria.
2. Rewrite draft descriptions so they fit the audience and documentation style.
3. Add missing context, examples, and workflow notes.
4. Check related links so readers can move to connected pages.
5. Send the page through technical review and writer review before publishing.

This is especially important for pages generated from route data, configuration keys, and exported items, because those pages often look complete at first glance even when they still lack the explanation readers actually need.

Fixing Common Problems in Parse-Driven Documentation Workflows

When parsed results in Atloria do not line up with what your team expects, start by checking scope before rewriting documentation. Missing items often come from incomplete parsing coverage rather than missing features. If a route, export, or configuration entry is absent, review whether the language is supported, whether the relevant folders were included in the parsing run, and whether generated or excluded files were filtered out. A missing result in the workspace usually means you should fix the parsing input before updating docs.

Relationship data can also be incomplete. Imports and dependency links are useful, but they may not capture every connection when a project uses aliases, indirect loading, generated files, or patterns that are harder to trace. If Atloria shows a relationship that seems too thin or misses an expected connection, treat the graph as a guide for review, not final proof. Confirm important links by opening the related parsed entries and checking the surrounding context.

Another common issue is noisy draft output. Not every parsed item belongs in published docs. Internal-only members, test files, deprecated exports, and private implementation details can clutter reference pages and make navigation harder. Before generating or approving drafts, filter the results to focus on reader-facing material. This keeps your technical docs useful instead of overwhelming.

Stale inputs are just as risky as missing ones. After merges, refactors, dependency changes, or route updates, rerun parsing in Atloria so your documentation work reflects the current project state. If a page still references an old file path or outdated signature, compare it against the latest parsing session and update the draft before publication.

When problems repeat, use the parsing workspace as a maintenance checkpoint. Review scope, rerun parsing, confirm relationships, and trim noise before your team spends time editing pages based on incomplete results.

Overview

Atloria’s Code Parsing workspace helps documentation teams turn raw repository analysis into usable documentation inputs. In this stage of the workflow, you are no longer just checking whether parsing succeeded. You are deciding how parsed results should support reference pages, project structure docs, setup guidance, and documentation planning across a project workspace.

The most useful outputs in Atloria are the ones that describe visible structure and reusable facts: exported items, file paths, route entries, imports, inline descriptions, and configuration keys. These results can speed up technical writing by giving you a reliable starting point for names, locations, relationships, and page groupings. They are especially helpful when you need to document large projects with many screens, connected features, and technical reference areas.

This guide focuses on how to use those results responsibly. Parsed data is strong at showing what exists and where it appears. It is not enough on its own to explain business purpose, user impact, release differences, or editorial meaning. That is why the workflow in Atloria combines parser-derived facts with writer review before anything is published.

Use this guide when you want to:

- Turn parsed results into draft reference pages
- Explain project structure using repository findings
- Identify documentation gaps from missing descriptions or uncovered routes

- Review parser-generated drafts before they move into published docs
- Clean up common issues such as missing items, noisy output, or stale parsing sessions

If you need help reading coverage and completeness first, return to Reviewing Parsed Code Results and Reference Coverage. The next step after this guide is Reviewing Code Parsing Results Symbols and Dependencies, where you will look more closely at how individual parsed items connect.

Prerequisites

Before you use parsed results to support technical docs in Atloria, make sure you already have a project workspace with code parsing results available to review. This guide assumes you are working from completed parsing output rather than starting a new upload.

You should be comfortable with these parts of Atloria before continuing:

- Opening a project workspace and accessing the Code Parsing area
- Reviewing parsing sessions and choosing the correct result set
- Checking coverage and completeness for the uploaded repository
- Navigating existing documentation pages in the same project
- Recognizing the difference between draft content and published content

It helps if you have already worked through these related guides:

- Uploading and Parsing Code in the Workspace
- Managing Code Parsing Workspace Sessions
- Using Code Parsing to Support Technical Documentation
- Uploading and Parsing Code for Documentation Workflows
- Reviewing Parsed Code Results and Reference Coverage

For the smoothest workflow, make sure:

- The repository you want to document has already been parsed in Atloria
- You are looking at the latest parsing session for the current project state
- Your team knows which documentation areas are in scope, such as reference pages, setup docs, or project structure pages
- You have permission to edit or plan documentation in the project workspace

If your next task is to inspect parsed items in more detail, continue with Reviewing Code Parsing Results Symbols and Dependencies.