ATLORIA

Reviewing Parsed Code Results and Reference Coverage

August 26, 2026

Opening a parsed code result and locating the readiness signals

After you upload code and run parsing, open the saved result from your project's Code Parsing workspace or from the connected repository view. If you need the upload steps, use Uploading and Parsing Code for Documentation Workflows. On the parsed result screen, start at the header area before opening any detailed panels. This is where Atloria shows the analysis name, the repository or source name, and the branch or revision label tied to that run. Use those labels first to make sure you are not reviewing an older snapshot or the wrong branch.

Next, look for the parse status indicator. Only review coverage after the result shows that parsing finished. If the status still shows that the run is in progress, queued, or incomplete, the files list, symbol list, and reference links may not represent the final result. The parse timestamp is equally important. When teams parse code more than once, the timestamp helps you confirm which run matches the version of the code you plan to document.

Once you confirm the correct snapshot, scan the main analysis panels. In Atloria, the most important areas for this review are the parsed file list, the symbols area, and any references or dependencies panel shown with the result. Coverage indicators beside the result tell you whether Atloria detected files, symbols, and cross-links between them. Those indicators are your first readiness signal: a result that shows files but little or no symbol or reference coverage usually needs closer inspection before you rely on it for technical writing.

[SCREENSHOT: Parsed code result header showing repository name, branch label, parse status, timestamp, and coverage indicators]

Inspecting files, symbols, and extracted structure

Once you have the correct parsed result open, move into the file tree or parsed file list. Your goal here is simple: confirm that Atloria included the source areas you

expected to document. Look for the main folders that matter to your project, such as app folders, shared code folders, integration areas, or documentation-related source directories. If an expected folder is missing from the file list, that is an early sign that your review may be incomplete.

After checking the file list, open the symbols panel. This view should show the named items Atloria extracted from the code. Depending on the codebase, you may see recognizable items such as classes, functions, methods, interfaces, or modules. You do not need to judge every item. Instead, spot-check a few important files and confirm that the symbols shown in Atloria match what you would expect from those files. If a major file appears in the file list but shows very few named items, the structural extraction may be weak.

Open a few symbol detail views and inspect the information Atloria provides. Useful signals include the symbol name, where it appears in the source, and whether it is grouped under a parent item or shown with child items beneath it. Those relationships help you judge whether Atloria understood the structure of the file instead of only listing isolated names.

A practical review pattern is to compare two or three high-value files against the parsed output. Check whether major items are present, whether grouped items are nested sensibly, and whether any names appear collapsed together in a confusing way. If you notice missing items or oddly grouped results in several important files, treat the parse as only partially reliable.

[SCREENSHOT: Parsed file list beside the symbols panel with one file expanded to show extracted structure]

Reviewing references and dependency links

After confirming that files and symbols were extracted, review whether Atloria captured relationships between them. Open the references or usages view for a symbol that you already know is used in more than one place. Good candidates are shared utilities, project-wide configuration items, reusable UI pieces, or common business logic. In a healthy parse result, these items should not appear isolated. You should see inbound and outbound links that help you understand where the item is used and what it depends on.

Next, inspect the dependency area for the parsed result. This may show links between files, modules, packages, or other grouped parts of the codebase. You are not trying to validate every connection. Instead, look for meaningful patterns. For example, if

one area of the project clearly relies on another, Atloria should show some visible relationship between them. When the dependency view is empty or unusually thin for a large codebase, that often means the parser captured structure but not enough cross-file context.

It also helps to open a symbol with known cross-file behavior and verify that related links appear. If you expect usage from other files, inherited behavior, imported dependencies, or implementation relationships, those should be visible in the references area when parsing is strong. Missing links do not always mean the parse failed completely, but repeated gaps reduce confidence.

Watch for broken patterns such as:

- empty usage lists for widely reused items
- missing links between obviously related files
- unresolved imports or disconnected dependencies
- symbols that appear in the list but have no surrounding context

When these patterns show up across several important areas, Atloria may still help with high-level structure, but it may not be strong enough for detailed behavior tracing.

[SCREENSHOT: Symbol detail view with references, usages, and dependency links visible]

Interpreting coverage indicators for documentation readiness

Coverage indicators in Atloria help you decide whether a parsed result is ready for documentation work or only useful as a partial reference. Start by separating two kinds of coverage in your review. The first is structural coverage: whether Atloria detected the expected files and extracted named items from them. The second is reference coverage: whether Atloria also connected those items through usages, dependencies, and other relationship links.

A result with strong file and symbol coverage but weak reference coverage can still be useful for some writing tasks. For example, it may support navigation, page outlines, component inventories, or high-level API summaries. By contrast, if you need to explain how a workflow moves across multiple files or how one part of the code triggers another, low reference coverage is a serious limitation.

As you review the indicators, pay attention to warnings that suggest likely documentation gaps. These may include low-coverage notices, unsupported-file

markers, or unknown-language labels. Those signals tell you where Atloria may have skipped extraction or produced only partial results. If those warnings appear in critical folders, you should avoid treating the parse as a complete source of truth.

Use the coverage view to decide what kind of documentation the result can support:

Documentation goal	Coverage you should look for
Architecture pages	Strong file coverage, clear structure, visible dependencies
API overviews	Strong symbol coverage, recognizable naming, stable grouping
Workflow documentation	Strong references, cross-file links, meaningful dependency paths

A parsed result does not need perfect coverage to be useful. It does need enough visible structure and linking to support the kind of page you plan to write without forcing constant manual verification.

Deciding whether the parsed output can support your documentation work

The best way to judge a parsed result is to compare it directly to your documentation goal. If you are drafting a high-level architecture page, focus on whether Atloria shows the major folders, major named items, and broad dependency relationships. If you are preparing API overviews, check whether the symbols list is complete enough to identify the public-facing surface of the code. If you are documenting a feature workflow, you need stronger references so you can follow behavior across multiple files with confidence.

Use three signals together when making that decision:

1. **Symbol completeness** — Do the important files show the main named items you expect?
2. **Dependency visibility** — Can you see meaningful links between major areas of the codebase?
3. **Reference density** — Do important symbols show enough usages and related links to trace behavior?

Partial parsing can still be useful. For example, if Atloria captured top-level project structure and major named items, you may be able to draft navigation pages, architecture summaries, or documentation outlines even when deeper cross-file references are incomplete. On the other hand, if your work depends on exact behavior tracing, sparse references should push you toward a narrower scope or a fresh parse.

It is also worth recording your review findings in a simple, repeatable way for your team. Note which folders look complete, which areas show weak coverage, and which documentation tasks are safe to start now. That gives writers, reviewers, and project owners a shared understanding of what the parsed result can support today and what still needs attention before drafting begins.

[SCREENSHOT: Coverage summary with notes or review comments captured beside the parsed result]

Handling common gaps in parsing results

When parsing results look incomplete, start with the most visible gap and work backward from there. If expected files are missing, first confirm that you opened the right repository source and the correct branch or revision label. In Atloria, a parsed result is tied to a specific snapshot, so reviewing the wrong branch can easily look like a parsing problem when it is really a scope problem. Also check whether the upload or parsing setup excluded folders that contain important source files.

If the files are present but the symbols look thin or incomplete, the issue may be limited to extraction quality rather than repository scope. Pay attention to unsupported-language markers or unknown-file indicators in the result. Generated files and files with parsing problems can also reduce the amount of structure Atloria can show. In that case, the file list may look complete while the symbols panel still feels sparse.

When references or dependencies are weaker than expected, review whether the project structure is broad enough to require more context than the current parse captured. Multi-package repositories, shared libraries, and imported project settings can all affect how well cross-file links appear in the result. If those relationships are central to your documentation task, weak links are a sign to pause before drafting detailed workflow content.

If coverage remains low after these checks, choose one of three practical responses:

- re-run parsing on the correct branch or a cleaner source snapshot
- narrow the documentation scope to areas with stronger coverage
- supplement the parsed result with manual review in the source files

That decision should match the urgency and type of documentation you are producing. For broad summaries, partial coverage may be enough. For technical reference pages, workflow tracing, or dependency-heavy explanations, it is usually better to improve the parse first.

Overview

This page helps you review a completed code parsing result in Atloria and decide whether it is reliable enough to support technical documentation. The focus is not on running a parse. Instead, it is on reading the result you already have and checking whether the extracted files, symbols, references, and dependency links are strong enough for the kind of documentation you want to write.

You will use the parsed result screen to answer a few practical questions:

- Did Atloria finish parsing the correct repository snapshot?
- Were the expected files and folders included?
- Do the extracted symbols look complete and recognizable?
- Are references and dependency links present where you would expect them?
- Is the overall coverage strong enough for architecture, API, or workflow documentation?

This review matters because not every parsed result is equally useful. Some runs are excellent for high-level structure but weak for cross-file tracing. Others may capture the right files but miss important named items in key areas. By checking readiness signals before drafting, you can avoid writing documentation from an incomplete or misleading analysis snapshot.

This guide assumes you already know how to upload code and start parsing in Atloria. If you need that workflow, return to Uploading and Parsing Code for Documentation Workflows. If you need help managing repeated runs or switching between parsing sessions, see Managing Code Parsing Workspace Sessions.

The next document in this sequence is Using Code Parsing Results to Support Technical Docs, which focuses on turning a reviewed parse into practical writing work.

Prerequisites

Before you review parsed code coverage in Atloria, make sure the following are already in place:

- You can sign in to Atloria and open the project workspace that contains the code parsing result.
- A code parsing run has already been completed for the repository or uploaded source you want to review.

- You know which repository, branch, or revision you intend to document so you can verify the correct snapshot in the parsed result header.
- You have a clear documentation goal, such as an architecture page, API overview, feature workflow, or technical reference section.
- You are familiar with the earlier setup steps covered in Uploading and Parsing Code in the Workspace or Uploading and Parsing Code for Documentation Workflows.

It also helps to have a short list of high-value files or folders ready before you begin. That makes it easier to spot-check whether Atloria extracted the parts of the codebase that matter most to your documentation work.

Use this review when:

- you need to confirm whether a parse is trustworthy before drafting
- you want to compare multiple parsing runs and choose the strongest one
- your team needs to understand where documentation can begin immediately
- you suspect missing files, weak references, or incomplete dependency mapping

If your goal is not review but active writing, continue to Using Code Parsing Results to Support Technical Docs after you complete the checks in this guide.