ATLORIA

Reviewing Code Parsing Results Symbols and Dependencies

August 25, 2026

Opening a parsing result and identifying what was analyzed

When you return to a completed code parsing run in Atloria, start by opening the parsing result record from the Code Parsing Workspace for the project you are working on. If you already reviewed coverage in Reviewing Parsed Code Results and Reference Coverage, use that same result entry so you are checking symbols and dependencies from the exact same analysis.

1. Open your project in Atloria and go to the **Code Parsing Workspace**.
2. Find the parsing result you want to inspect in the results list or session history.
3. Open the result and read the header area first.
4. Confirm the visible status details, including whether the run completed successfully and when it was last analyzed.
5. Check the language, file type, repository, package, or source scope shown in the result summary.
6. Look for any indication that the run covered the full codebase or only selected files or folders.
7. Use the result navigation to identify the separate views for file coverage, symbols, and dependencies before you begin reviewing details.

The header is the fastest way to avoid reviewing the wrong run. If the result shows only a limited selection of files, your symbol list and dependency map may be incomplete by design. That matters when you are planning architecture pages, API reference sections, or feature explanations.

Pay close attention to how Atloria separates what was scanned from what was extracted. A file coverage area tells you what source content was included. A Symbols view shows what named items Atloria recognized inside that content. A Dependencies or Relationships view shows how those items connect. These are not interchangeable, so it helps to confirm which tab or panel you are on before drawing conclusions.

[SCREENSHOT: parsing result page showing header details, analysis timestamp, scope summary, and tabs for files, symbols, and dependencies]

Reviewing extracted symbols to see what the parser understood

The Symbols view is where you check whether Atloria recognized the important named parts of the code you plan to document. This is especially useful when you need to confirm whether public-facing areas were captured clearly enough to support technical writing.

1. Open the **Symbols** tab or symbols panel inside the parsing result.
2. Scan the symbol list for recognizable names tied to the feature or code area you are documenting.
3. Review the symbol type shown for each entry, such as class, function, method, interface, module, or constant.
4. Use the file path or source location shown beside each symbol to confirm where it comes from.
5. Open a few key symbols to inspect their details.
6. Compare top-level items with nested items to see whether Atloria preserved parent-child structure.
7. Note any important gaps, such as missing public entry points or symbols with unclear names.

A strong symbol list usually includes meaningful names, clear symbol types, and enough location detail to connect the item back to the source file. If you open a symbol detail panel and see information such as its signature, visibility, parent item, or source range, that gives you more confidence that Atloria understood the structure rather than only listing text matches.

Look closely at relationships inside the symbol list. For example, if a top-level item contains several nested methods or functions, that suggests Atloria preserved the internal structure of that file. If everything appears flattened into one long list, you may need to be more cautious when using the result to explain ownership, feature boundaries, or public interfaces.

This is also the point where you can separate likely documentation priorities from lower-value details. Publicly exposed items, shared modules, and clearly named entry points are usually more useful than deeply nested internal helpers.

[SCREENSHOT: Symbols tab showing symbol names, types, file paths, and an expanded symbol detail panel]

Inspecting dependencies and relationships between files and symbols

After reviewing symbols, switch to the Dependencies or Relationships view to understand how Atloria connects files and named items. This view helps you move beyond “what exists” and into “what depends on what,” which is often the basis for architecture explanations, integration notes, and technical diagrams.

1. Open the **Dependencies** or **Relationships** tab in the parsing result.
2. Select a file or symbol that appears important in your documentation plan.
3. Review the linked entries that show what it imports, exports, references, extends, calls, or connects to.
4. Follow one relationship at a time into the next linked file or symbol.
5. Compare direct links with broader relationship chains so you can tell immediate dependencies from indirect ones.
6. Watch for missing or surprising gaps in the relationship list.
7. Record only the connections that Atloria shows clearly and consistently.

When you trace a relationship, stay grounded in what the screen actually shows. A direct dependency is a visible first-level connection between one file or symbol and another. A broader chain may involve several hops. For documentation, that difference matters. If Atloria shows File A linked to File B, and File B linked to File C, do not automatically describe File A as directly dependent on File C unless the relationship view makes that explicit.

Missing links can be just as informative as visible ones. If a public entry point appears isolated, or a central shared module has no incoming references, that may mean the parse is incomplete, the code area was only partially included, or the relationship extraction was limited for that language or file type.

Use this view to identify likely feature entry points, shared building blocks, and cross-file interactions. Those patterns are often more useful for documentation planning than a raw list of files.

[SCREENSHOT: dependency view showing linked files or symbols with relationship labels and selectable rows or graph nodes]

Judging whether the parsed output is useful for documentation

Not every parsing result is strong enough to support documentation decisions. Before you rely on it, use the visible evidence in Atloria to judge whether the output is complete and accurate enough for the kind of writing you need to produce.

1. Compare the result against the documentation task you are working on.
2. Check whether the files you need are present in the parsed scope.
3. Confirm that the main symbols for that area appear in the Symbols view.
4. Verify that the Dependencies or Relationships view shows the key connections you expected.
5. Open a few known files or code areas and compare what Atloria extracted with what you already know from the project.
6. Decide whether the result is good enough for planning, drafting, or only limited reference support.
7. Mark any areas that still need manual review before you write confidently.

For architecture pages, you usually need broad coverage across folders, recognizable entry points, and visible relationships between major code areas. For API overviews, you need clear public symbols and stable naming. For feature explanations, you need enough structure to identify where a workflow starts and what supporting pieces it touches.

Warning signs are usually easy to spot once you look for them:

What you see in Atloria	What it may mean for documentation
Many unnamed or generic symbols	The parser did not capture enough structure
Missing expected files or folders	The parse scope may be partial
Sparse or broken relationships	Dependency extraction may be incomplete
Unsupported or mixed file types	Coverage may vary across the codebase
Very limited symbol details	Use the result cautiously for deeper explanations

If the result is only partially reliable, you can still use it for orientation and topic discovery. Just avoid turning weak signals into firm statements in published documentation.

Using parsing results to support technical writing decisions

Once you trust the parsing result enough, you can turn what Atloria shows into practical writing decisions. The goal is not to repeat the code structure word for word, but to use symbols and relationships to shape documentation that matches the reader's needs. If you need a broader workflow for using parsed output in writing, continue to rely on Using Code Parsing Results to Support Technical Docs.

1. Review the strongest symbols and relationship clusters in the parsing result.
2. Group related symbols into possible documentation topics or section headings.
3. Identify which items look publicly visible or externally important.
4. Use dependency links to locate feature entry points and shared supporting areas.
5. Separate internal implementation details from items that deserve reader-facing coverage.
6. Flag unclear areas that need source review or team confirmation.
7. Turn your findings into a documentation outline before drafting.

A symbol list often maps naturally to topic planning. Exported modules, clearly named interfaces, and public classes can become candidate headings in a reference page or technical overview. Shared utilities and cross-cutting dependencies may deserve a short explanation, a diagram, or a note about reuse across features.

Dependencies are especially helpful when you need to explain flow. If several files point back to one common entry point, that may be the right place to anchor a feature explanation. If one shared area appears across many relationships, it may deserve its own section so readers understand why it matters.

Keep your audience in mind. Not every extracted symbol belongs in user-facing technical documentation. Focus on stable, externally meaningful names and visible boundaries. When Atloria shows incomplete relationships or ambiguous naming, treat those areas as research prompts rather than final answers.

[SCREENSHOT: parsed results used alongside a documentation outline or notes panel for planning topics]

Handling incomplete or confusing parsing results

Sometimes the parsing result in Atloria looks thin, inconsistent, or harder to trust than expected. When that happens, work through the visible scope and result details before deciding whether to re-run the parse or validate the code manually.

1. Reopen the parsing result header and confirm the repository, package, branch, or file selection that was analyzed.

2. Check whether the run covered the full project or only selected folders or files.
3. Review the language or file type shown in the result details.
4. Return to the Symbols view and confirm whether missing items are absent everywhere or only in one area.
5. Open the Dependencies or Relationships view and look for signs that links were only partially resolved.
6. Decide whether the issue is likely caused by limited scope, weak parser support, or a genuinely simple code area.
7. Re-run the parse when the wrong content was included, and use manual validation when the result is present but unclear.

A few patterns can help you decide what to do next:

- If expected files are missing, the parse likely did not include the right source scope.
- If names look collapsed, overly generic, or incomplete, the language support for that file type may be limited.
- If dependencies are sparse even though symbols appear correctly, the relationship extraction may not have resolved fully.
- If only one folder looks weak while the rest of the result looks strong, you may be dealing with a localized gap rather than a failed run.

Use re-parsing when the problem is about what Atloria included. Use manual source review when Atloria included the right area but the extracted structure is still too thin to support confident documentation. That distinction saves time and keeps your documentation planning realistic.

Overview

This page focuses on one specific task in Atloria's Code Parsing Workspace: reviewing a completed parsing result closely enough to decide whether it can support technical documentation work. You are not uploading code here, starting a new workspace session, or checking general parser coverage. Those tasks are covered elsewhere, including Uploading and Parsing Code in the Workspace, Managing Code Parsing Workspace Sessions, and Viewing Supported Languages and Parser Coverage.

The main areas to review in Atloria are:

- The parsing result header, where you confirm status, scope, language or file type, and analysis timing
- The **Symbols** view, where you inspect the named items Atloria extracted from the source
- The **Dependencies** or **Relationships** view, where you trace how files and symbols connect
- The visible gaps or warnings that affect whether the result is reliable enough for documentation planning

This review is most useful when you are preparing technical overviews, API reference coverage, architecture notes, or feature explanations and need to know whether the parsed output is trustworthy. A good result can help you identify entry points, shared areas, and documentation priorities. A weak result can still help with orientation, but it should not be treated as complete evidence.

The next document in this sequence is Uploading Code and Reviewing Parsing Results, which brings the upload and review steps together into one workflow.

Prerequisites

Before you review symbols and dependencies in Atloria, make sure you already have the right project context and at least one completed parsing result available.

- You can open the relevant project and access its **Code Parsing Workspace**
- A parsing run has already finished for the repository, package, or source files you want to inspect
- You know which code area you are trying to document, such as a feature area, API surface, or architecture boundary
- You have already reviewed general result coverage in Reviewing Parsed Code Results and Reference Coverage
- You understand the writing goal well enough to judge whether you need broad architecture coverage, public symbol coverage, or dependency mapping
- You are ready to compare what Atloria shows against a few known code areas when needed

It also helps if you have already worked through Using Code Parsing Results to Support Technical Docs. That earlier guide explains how parsed output fits into documentation work. This page stays narrower: it shows how to inspect the result

itself and decide whether the extracted symbols and relationships are dependable enough to use.