ATLORIA

Uploading Code and Reviewing Parsing Results

August 25, 2026

Opening the code parsing workspace

In Atloria, open the **Code Parsing Workspace** from your project's technical documentation area, then look for the four parts you will use most during a parsing run:

- the **upload panel**, where you add code files
- the **code snippet editor**, where you paste short samples
- the **parse** or **run** controls, where you start analysis
- the **results panes**, where Atloria shows symbols, dependencies, and the parsing summary

The upload panel is best when you want Atloria to analyze real project files together. This is usually the right choice if you are checking how files relate to each other or if you want more complete symbol and dependency results. The code snippet editor is better for quick tests, such as checking whether a small block of code is recognized before you upload a larger set of files.

Before you start a run, confirm which content Atloria should analyze. If you are using files, make sure the correct files appear in the upload area. If you are using the snippet editor, review the pasted code carefully and remove anything unrelated. Then use the main **Parse** or **Run** action to begin.

After the run finishes, move through the results panes one by one. The **symbols list** shows the definitions Atloria detected in the uploaded content. The **dependency view** helps you see which files or code elements connect to each other. The **parsing summary** gives you the overall outcome, including whether the run completed successfully and how much Atloria recognized.

If you already know how to read symbol and dependency details, this page focuses on the full upload-and-review workflow rather than repeating that material from Reviewing Code Parsing Results Symbols and Dependencies.

[SCREENSHOT: Code Parsing Workspace showing upload panel, snippet editor, parse button, and results panes]

Preparing files and snippets for parsing

Atloria can work with several kinds of source material in the **Code Parsing Workspace**:

- a single code file
- multiple related code files
- a short pasted snippet in the editor

Choose the format that matches your goal. If you want a quick parser check, paste a short example into the snippet editor. If you want to understand relationships across a feature or section of a project, upload multiple related files together so Atloria can review them as one set.

File names and file extensions matter because they help Atloria identify the language and structure of what you upload. A clearly named file with the correct extension usually gives better results than a renamed file or a pasted fragment with no surrounding context. Snippets can still be useful, but they work best when they include complete, readable code rather than isolated lines.

Before you upload or paste content, check a few basics:

- remove broken or unfinished fragments if possible
- make sure the code matches the language you intend to analyze
- keep related files together instead of uploading only one file from a larger connected group
- avoid mixing unrelated samples in the same run when you want clear results

For snippets, include enough surrounding code for Atloria to recognize the structure. For files, try to upload the actual source files rather than copied text saved under the wrong extension.

If you manage a shared workspace, also confirm that you can access the project area where parsing is available. Project administrators should verify that the right people can open the workspace and that any file-type or upload-size rules used by their Atloria setup are understood before team members begin. If a file cannot be added, check those limits first instead of assuming the parser failed.

[SCREENSHOT: File selection area with multiple uploaded files listed before parsing]

Uploading code and starting a parsing run

1. Open the **Code Parsing Workspace** in Atloria and go to the **upload panel** if you want to analyze files.
2. Use the upload control to add one or more source files. After you select them, check the file list shown in the workspace. Make sure the files you expected to upload are all present and that you did not include unrelated files by mistake.
3. If you only want to test a small sample, switch to the **code snippet editor** instead of uploading files. Paste the code directly into the editor and review it for missing lines, cut-off sections, or extra text that is not part of the code.
4. Before starting the run, review the content shown on screen:
    - confirm the correct files appear in the upload list, or
    - confirm the snippet editor contains the exact sample you want Atloria to parse
5. Click the main **Parse** or **Run** action to start analysis.
6. Watch for any in-progress status shown in the workspace. Atloria may display a loading state, a running indicator, or a temporary status message while it processes the selected files or snippet.
7. Wait for the results panes to update before reviewing the output. Starting another run too quickly can make it harder to tell which results belong to which upload.

If you are comparing parser behavior, run one clean test at a time. For example, first upload a related file set and review the results, then try a smaller snippet separately. That makes it easier to see whether missing symbols or dependencies are caused by the content itself or by the way it was submitted.

[SCREENSHOT: Parse button selected with upload list visible and run status indicator in progress]

Reviewing symbols and dependencies in the results

1. Start with the **symbols** panel. This area lists the definitions Atloria detected in the uploaded files or pasted snippet. Depending on the code you submitted, you may see items such as classes, functions, methods, or other named definitions.
2. Scan the list for the main items you expected Atloria to find. If an important definition is missing, compare the results with the original file or snippet you submitted.

3. Open the **dependency** view next. This area shows relationships between uploaded files, referenced modules, or linked code elements that Atloria recognized during the run.
4. Compare the dependency view with your upload set. If you uploaded several related files, you should be able to see whether Atloria connected them. If the dependency view looks sparse, that often means the run did not include all related files.
5. Return to the original uploaded content and verify that the structures shown in the results match what is actually in the source material.

For technical writers and documentation managers, these two result areas are especially useful when planning coverage. The symbols panel helps you identify what named items exist and may need documentation. The dependency view helps you understand which files or features are connected, which is useful when you are mapping a documentation section to a real part of the codebase.

If you need a deeper explanation of how to interpret individual symbol entries and relationship details, use Reviewing Code Parsing Results Symbols and Dependencies. In this workflow, the goal is to confirm that your upload produced a usable set of results before you move into documentation work.

[SCREENSHOT: Results area with symbols list on one side and dependency view on the other]

Interpreting parsing summaries and run outcomes

After reviewing symbols and dependencies, check the **parsing summary** to understand the overall run. This summary is the fastest way to confirm whether Atloria processed the content as expected. Look for totals such as:

- files processed
- symbols detected
- dependency counts
- overall run status

A successful run usually shows a completed status and meaningful totals in the summary. If you uploaded several files and Atloria reports multiple symbols and dependencies, that is a strong sign that the parser recognized the structure of your content.

Partial results are different. In that case, Atloria may still show some symbols or relationships, but the summary may indicate warnings, incomplete processing, or fewer detected items than expected. This often happens when one file in the upload set is incomplete, unsupported, or missing related context.

A failed run is usually easier to spot because the summary area will show an error state or a clear failure message. When that happens, review any warning or error messages attached to the file list, snippet area, or summary panel. These messages often point to the exact file or pasted sample that caused the problem.

Use the summary to decide what to do next:

- re-run with corrected files if one upload was incomplete
- test a smaller snippet if you want to isolate a problem quickly
- upload a more complete related file set if dependencies or symbols look too limited
- remove unrelated files if the run included too much mixed content

The summary is not just a status check. It helps you judge whether the results are complete enough to support your next documentation task, which you will build on in Using Code Parsing Results to Improve Technical Documentation.

[SCREENSHOT: Parsing summary showing totals, run status, and warning messages]

Fixing common upload and parsing problems

When a parsing run does not give the results you expected, start with the message shown in the workspace and then check the content you submitted.

If file upload does not start, review the basics first:

- confirm the file type is allowed in your Atloria workspace
- check whether the file may be too large
- make sure you have access to use the Code Parsing Workspace in that project

If the file never appears in the upload list, the issue is usually with the file itself or workspace access rather than the parsing step.

If a snippet parses with missing symbols, look closely at what you pasted into the **code snippet editor**. Missing symbols often come from incomplete code blocks, cut-off declarations, or samples that depend on surrounding code that was not included. In that case, paste a fuller example or upload the actual file instead.

If dependencies look incomplete, the most common cause is analyzing files in isolation. Atloria can only show relationships it can see in the uploaded set. If you upload one file from a larger feature, the dependency view may miss links that would appear if the related files were included in the same run.

If the parsing run fails or returns warnings, read the run status message in the **parsing summary** and check whether a specific file or snippet is named. Then retry after correcting that item. A smaller test run can help you confirm whether the issue is limited to one file.

When troubleshooting, change one thing at a time. For example:

- first retry with the same files after removing one problematic file
- then test a short snippet from that file
- then upload the full related set again

That approach makes it easier to see what fixed the problem.

Overview

This page walks through the complete basic workflow for using the **Code Parsing Workspace** in Atloria: adding source material, starting a parsing run, and reviewing the results that matter most for documentation work. The focus here is practical use inside the workspace rather than parser theory.

You will work with four main areas in the screen:

- the **upload panel** for adding files
- the **code snippet editor** for quick pasted samples
- the **Parse** or **Run** action for starting analysis
- the results area for **symbols**, **dependencies**, and the **parsing summary**

Use this workflow when you want to answer questions such as:

- Did Atloria recognize the main definitions in these files?
- Are related files connected in the dependency view?
- Did the run complete successfully, partially, or with errors?
- Is this result set strong enough to support technical documentation work?

This guide does not repeat the deeper interpretation of symbol and dependency details already covered in Reviewing Code Parsing Results Symbols and

Dependencies. Instead, it shows how to move from raw source material to a completed parsing run you can trust.

For most teams, the best results come from uploading a clean, related set of files and then using the summary to confirm that Atloria processed them correctly. Short snippets are still useful, especially when you want to test a parser quickly or isolate a problem before running a larger upload.

Once you are comfortable with this workflow, the next step is using these parsing results to shape clearer documentation structure, coverage, and reference planning in Using Code Parsing Results to Improve Technical Documentation.

Prerequisites

Before you begin in Atloria, make sure the following are in place:

- You can open the **Code Parsing Workspace** in the project where you want to work.
- You have source material ready to submit, such as:
  - a single code file
  - several related files
  - a short snippet for testing
- Your files use the correct names and extensions so Atloria can identify them more accurately.
- Your snippet, if you are using one, is complete enough to show the structure you want Atloria to detect.
- You know the intended language or framework of the content you are submitting.
- If you are uploading multiple files, you have grouped related files together instead of mixing unrelated samples.
- You have already reviewed the earlier parsing guidance if you need background on workspace setup or result interpretation:
  - Uploading and Parsing Code in the Workspace
  - Managing Code Parsing Workspace Sessions
  - Uploading and Parsing Code for Documentation Workflows
  - Reviewing Parsed Code Results and Reference Coverage

If you are working in a shared team space, project administrators should also confirm that team members have the right workspace access and understand any file-type or

upload-size limits used in that Atloria project. That helps avoid failed uploads before parsing even begins.

For the smoothest first run, prepare one clean upload set or one clean snippet rather than testing several unrelated samples at once.