

The Tool

This page is the second proof of the blueprint’s premise: it is authored in MyST, transpiled to MDX by `myst2mdx` — the exact tool it describes — and rendered by Starlight below. [The blueprint](#) shows the pipeline as a whole; this page shows the piece that makes it work.

1. AN ORCHESTRATOR, NOT A DOCUMENTATION ENGINE

`myst2mdx` does not parse MyST — `mystmd` does that upstream, before this page’s AST ever reaches the transpiler. It does not render generic Markdown either — Astro/MDX does that downstream. It owns exactly one thing: the policy of how a *resolved* MyST AST node maps onto a Starlight or MDX construct.

Note

“Resolved” is load-bearing. Cross-references, numbering, and the table of contents are already final by the time `tool/` sees this page’s AST — `myst build --site` computed them upstream. The transpiler’s own cross-reference handling is a single plain `<a href>`, not a runtime resolver.

2. NATIVE WHERE POSSIBLE, FALLBACK WHERE NOT

A fixed set of MyST node types — paragraphs, headings, lists, links, code, admonitions, images, and cross-references among them — render as real Starlight/MDX constructs. Everything outside that set still renders, just through an HTML fallback channel that emits JSX-valid markup instead of breaking the build.

Tip

Extending native support for a new node type means two changes, not one: adding the type to the tool’s native set *and* teaching the emitter what MDX or Starlight construct it becomes. Skip the second half and the type silently falls back to HTML instead of failing loud.

3. FOUR ESCAPING CHANNELS, NOT ONE

MDX generation breaks on a stray `{`, `<`, or backtick reaching the compiler unescaped. The tool routes every character through one of four channels, depending on where it lands:

- prose text — neutralised (`{`, `}`, `<`, backtick)
- code fences — passed through verbatim, fence length auto-sized to the content
- component attribute values — entity-escaped
- the raw-HTML fallback — JSX-valid markup: self-closing void tags, quoted attributes

Danger

None of this is trusted blindly. `astro build` — the real MDX compiler — is the actual correctness gate; the escaper's only job is to make that gate pass.

4. ONE CORE, TWO TARGETS

The transpiler's logic lives once, in a `commonMain` Kotlin source set, and compiles to two targets: a JVM CLI that walks a directory of AST JSON files (what built this page), and a `@JsExport`'d `transpileToMdx()` function for the JS target — the same `transpile()` entry point, same behavior, callable from JavaScript, by construction rather than a second implementation kept in sync by hand. No browser UI consumes it yet; today it's a proven, exported API surface, not a shipped playground.

5. ADMONITIONS, COLLAPSED

MyST has more admonition kinds than Starlight has `<Aside>` types. They collapse through a table the tool owns. This section demonstrates the full collapse, live — what you see rendered below is that table in action:

Note

`note`, `important`, `hint`, `seealso`, and the bare admonition kind all collapse to this same note aside.

Tip

`tip` keeps its own aside type.

Warning

`warning`, `caution`, and `attention` all collapse to this same caution aside.

Danger

`danger` and `error` both collapse to this same danger aside.

Eleven admonition kinds, four aside types — the mapping lives in `NodeMapping.ASIDE`, not in this prose, so it can't drift out of sync with what just rendered above it.

6. NEXT

[Browse the generated API reference `arrow.r`](#) — every public declaration in `tool/`, transpiled from its own KDoc comments by the same `dokka2mdx` subcommand described above, not hand-written.