Cupertino v1.1.0: my Apple docs index was 30% lies and I didn't know

`brew upgrade cupertino` this week and your local Apple docs database is 7,095 documents richer and zero rows are HTTP error pages pretending to be documentation. You will not notice. Searches feel the same. `cupertino doctor` was already green. The MCP server returns the same shape.

That gap, between "looks fine" and "actually fine", is the whole post.

TL;DR

```
brew update
brew upgrade cupertino
cupertino setup     # downloads cupertino-databases-v1.1.0.zip (685 MB)
cupertino doctor    # "Bundled version: 1.1.0", apple-docs: 284,518
```

Done. The rest is how I found out the index was lying to me.

The 3 a.m. query

I was confident about v1.0.2. Audit clean. Bundle rebuilt. Counts verified. Shipped.

Then I ran one more query against the v1.0.2 `search.db`, the kind you only run when you can't sleep:

```
SELECT uri, title, abstract, source
FROM docs_metadata m JOIN docs_structured s ON s.uri = m.uri
WHERE s.title IN ('Forbidden', '403 Forbidden', '404 Not Found',
                  'Bad Gateway', '502 Bad Gateway');
```

One row came back.

```
uri:      swift-org://docc_documentation
title:    Forbidden
abstract: You don't have permission to access this resource.
source:   swift-org
```

That is not Apple's DocC documentation. That is an HTTP 403 page indexed as canonical content for the DocC URL. Anyone querying cupertino for DocC would get a result with a correct URL, a plausible title, and a body of pure garbage. The worst kind of indexing bug: silent and authoritative.

The mistake was small and embarrassing. My poison scan walked `~/.cupertino/docs/`. The bundle is built from **six** sibling directories. I had been auditing one out of six.

```
~/.cupertino/
  docs/             # 412k JSON files
  swift-evolution/  # 483 proposals
  swift-org/        # 196 pages    ? never scanned
  hig/              # 173 pages
  archive/          # 406 legacy guides
  packages/         # README files
```

Sweeping the full tree turned up six poison files in `swift-org/`. Five of them had been shipping in every prior release. Nobody had noticed. But the six was the small problem. The big problem: my scanning was wrong, and I had no idea what else was hiding in 412,523 other files.

The 13 ways Apple's docs site poisons a crawler

I rebuilt the audit as a matrix: every pattern, every source, every file. Each row below came from a real failure I had already hit. None of it is theoretical.

#	Pattern	What it actually is
1	Title `403 Forbidden`, `502 Bad Gateway`, `404 Not Found`, `500`, `503`, `504`, `429`	Apple's CDN serves an HTTP error page at status 200, gets indexed as a doc page
2	Title bare phrase: `Forbidden`, `Bad Gateway`	Same as 1 minus the status code
3	Body: `Please turn on JavaScript in your browser…`	React SPA noscript fallback
4	Body: `[ Skip Navigation ](#app-main)# An unknown error occurred.`	React app's generic error sub-view rendered as the whole body

#	Pattern	What it actually is
5	Body contains `openresty`	Apple's edge CDN error template
6	Body contains `<center>nginx`	Different nginx error template
7	Body: `You don't have permission to access this resource`	Forbidden as plain prose
8	Body: `cf-error-code` / `Cloudflare Ray ID`	Cloudflare error page
9	Body: `Reference&#32;&#35;` / `Akamai` markers	Akamai edge error
10	Body: `<Code>NoSuchKey</Code>` / `<Code>AccessDenied</Code>`	AWS S3 XML errors for missing assets
11	Zero-byte files	Truncated writes from killed crawls
12	Files under 200 bytes	Almost-empty responses
13	`.json` files not starting with `{` or `[`	Corrupt JSON, sometimes HTML in disguise

Against every expectation, `docs/` scanned clean across all 13. Earlier passes had actually stuck. The poison was concentrated in the small directories I had been ignoring.

(Quick footnote, since it cost me an hour. Category 13 is a byte read on 412k files. My first pass shelled out `head -c 1` per file. Projected wall time: about five hours. Rewriting as a single Python `os.walk` that stats, opens, and reads one byte inside one process: 41 seconds. The per-file fork was the entire cost.)

How Apple's docs site actually poisons your index

Three failure modes. In the order they fooled me.

Mode 1: the CDN returns an HTML error page at status 200. Title is `403 Forbidden`. A five-line title check catches it. Done in PR #289 last release.

Mode 2: the SPA shell. Apple's docs site is a client-rendered React app. Fetch without JS and you get a `<noscript>` blurb that politely asks you to enable JavaScript. v1.0.2 had 1,327 of these. PR #291 filtered them at indexing time, but it was scoped to the apple-docs path, which is exactly why the `swift-org` `Forbidden` row strolled past.

Mode 3: the one that ate my Tuesday. JavaScript runs. The React app boots. The React app's data API returns 404 for the URL. The React app cheerfully renders its own 404 sub-view as the page body:

```
<h1>Apple Developer Documentation</h1>
<a href="#app-main">Skip Navigation</a>
<h1>The page you're looking for can't be found.</h1>
<input placeholder="Search developer.apple.com">
```

Title is generic. Body is the app announcing its own failure. No error markers. Status 200. HTML perfectly clean. Every defense I had waved this through, because from the network's point of view Apple successfully served me a page. The page just happens to be Apple's React app politely informing the user that the page doesn't exist.

The fix that was missing

PR #432 is small. A substring check on the rendered HTML, before the crawler writes anything:

```
public static func looksLikeJavaScriptFallback(html: String) -> Bool {
    if html.contains("The page you're looking for can't be found") { return
true }
    if html.contains("An unknown error occurred") { return true }
    return false
}
```

Both phrases live exclusively inside Apple's no-content sub-views. Real docs do not quote either sentence. The check runs after WebKit renders, so it sees the same bytes that would have hit disk. If it returns true, the crawler logs a skip and walks away.

Every skip writes one JSONL row to `<output>/.cupertino-rejected-urls.jsonl`:

```
{"url":"https://developer.apple.com/documentation/.../fetch-apple",
 "framework":"accountorganizationaldatasharing",
 "reason":"js_fallback",
 "timestamp":"2026-05-13T11:53:42Z"}
```

This is the part I think generalizes. Once a poison file lands on disk, every downstream consumer has to re-discover that it's poison. Catch it at write time and it is nobody's problem ever again. The JSONL puts the visibility back: `jq` over it and you have a retry list, a recrawl set, or a bug-report folder.

Targeted recrawl of 51 known-poison URLs from v1.0.2: 48 rejection rows, all correctly tagged `js_fallback`, zero new files on disk. Pre-fix, the same set wrote 9 poison files in 90 seconds before I killed the run.

One false-positive risk worth naming. Two real Apple symbol pages quote "An unknown error occurred" in their Discussion section (`cfnetworkserviceerrorunknown`, `generalinternalretryableerror`). Both are already in the v1.1.0 bundle, but a future recrawl of those exact URLs would currently get rejected. Filed for v1.2: tighten the match so the phrase has to sit in a standalone position (h1, adjacent to Skip Navigation) rather than anywhere in the HTML.

The reindex

`cupertino save --docs --clear` on the cleaned corpus:

```
documents:          285,735       (84.7% of 351,249 source, rest deduped)
frameworks:         420
db size:            2.36 GB
defense trips:      0
errors:             0
warnings:           0
runtime:            9h 39m
```

The line I care about is `defense trips: 0`. Across 351,249 indexer reads the indexer-side gate from PR #291 did not fire once. The test wasn't whether the defense works. The test was whether anything was left for it to filter. Nothing was.

(The 65,514 gap between source files and indexed rows is real deduplication. Apple's docs ship

hash-suffixed variants of overloaded methods (`equatable_-3axv1_de895d6c.json`) that all point at the same logical URL.)

The release saga, or: what `databaseVersion` actually does

This is the part I'd flag to anyone shipping a similar tool.

v1.1.0 first shipped tagged correctly, binary rebuilt correctly, bundle uploaded to a parallel GitHub release. I ran `cupertino setup` to test the upgrade. It cheerfully downloaded the **old** v1.0.2 bundle.

The bug was one line in `Shared.Constants.swift`:

```
public static let databaseVersion = "1.0.2"
```

`cupertino setup` builds the download URL from this constant:

```
{baseURL}/v{databaseVersion}/cupertino-databases-v{databaseVersion}.zip
```

The release-tooling commit had explicitly not bumped it, with a confident note: "databaseVersion stays at 1.0.2 (no schema/content change)." Schema-wise that was true. Content-wise it was a confident lie. The corpus was the cleanest it had ever been, the bundle was genuinely new, and the binary was politely asking the CDN for last week's poisoned one.

Fix: one-line change, fix branch, squash-merge, force-move the v1.1.0 tag to the new HEAD. `release.yml` rebuilds, codesigns, notarizes, and replaces the assets on tag push. Then a Homebrew bump for the new tarball SHA. About 25 minutes of clock time once I noticed. Approximately 25 minutes of dignity, also gone.

The lesson: when the bundle changes, bump `databaseVersion`, even if the schema doesn't. Otherwise your install path silently downgrades users to the previous bundle. No error. Doctor reports green. The binary asked for last week's bundle, and last week's bundle is exactly what it got.

What I'd take with me

HTTP 200 doesn't mean "valid content." It means the CDN served some bytes. Documentation, error page, noscript fallback, React 404 sub-view: your problem to tell apart. Title checks miss two of three.

Defense gates belong upstream. Catching poison at write time costs the same as catching it later, and prevents a class of bug instead of detecting one. Append an audit log so you keep the visibility.

Bump `databaseVersion` when the bundle changes, even if the schema doesn't. Then run `cupertino setup` end-to-end before the release is "shipped." Trust the binary, not the release notes.

If you want to try it:

```
brew install mihaelamj/tap/cupertino    # or brew upgrade
cupertino setup
cupertino search "your favorite Apple symbol"
```

Three defense PRs: #289, #291, #432. About 350 lines of Swift including tests. The cleanup itself was one long afternoon of careful greps.