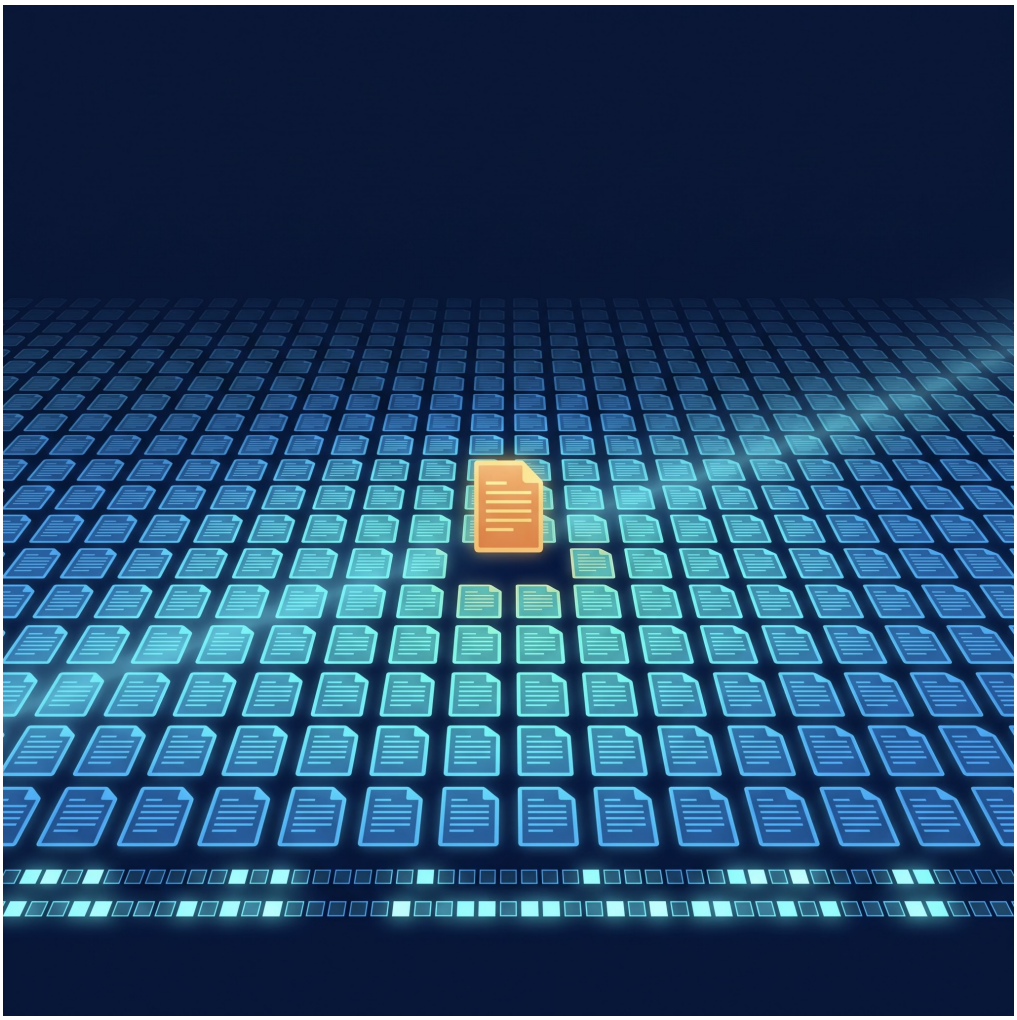
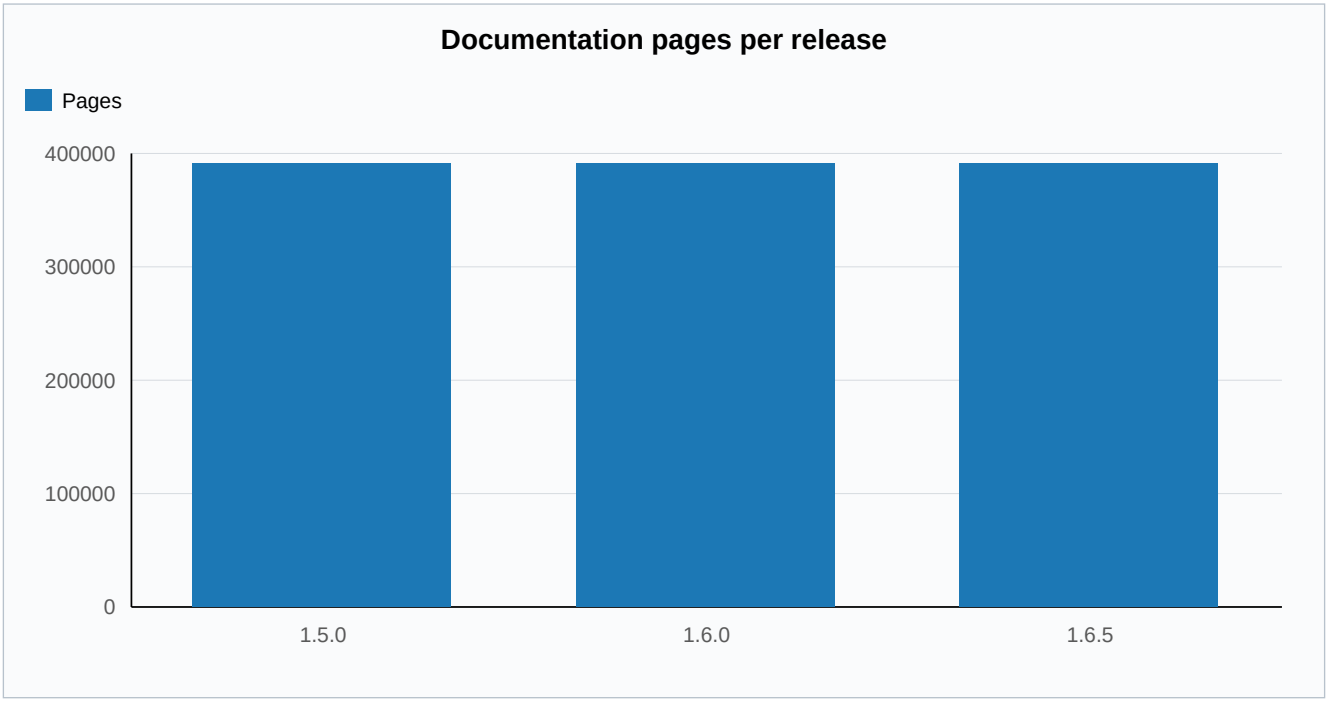


# Cupertino 1.6: Platform Filters, Apple's Tutorials, and One Search Engine

Cupertino 1.5.0 was about the content: 27,494 new pages and not one lost. The two releases since then, 1.6.0 on 18 November and 1.6.5 on 19 November, are about asking better questions of that content, adding a kind of page the database never had, and fixing the places where a search did something it should not.

The short version: **you can now ask for pages that exist on exactly the platforms you name, or for one UI toolkit. Apple's tutorials are searchable. A search no longer changes the files it reads. And the code that reads the databases is on its way to becoming one implementation instead of two.**

The long version follows, with the numbers, because the numbers are where the mistakes were found.




The page count barely moves. That is the point: 1.6.0 changed what is stored about each page, and 1.6.5 added 193 pages of a new kind. Every one of the 391,056 pages of 1.6.0 is unchanged in 1.6.5, which I checked row by row, and I will show how.

## Filtering by platform: `--only`

Until 1.6.0 you could say "available since iOS 17" (`--min-ios 17.0`) or "not after macOS 14" (`--max-macos 14.0`). You could not say "this exists on iOS and visionOS and nowhere else". Issue #8 asked for that.

The answer is a new column in schema 20, `docs_metadata.platform_scope`, a bitmask of the platforms a page's availability list names:

| Platform | Bit | Value |
|---|---|---|
| iOS | 0 | 1 |
| macOS | 1 | 2 |
| tvOS | 2 | 4 |
| watchOS | 3 | 8 |
| visionOS | 4 | 16 |
| Mac Catalyst | 5 | 32 |

A page's scope is the sum of the values of its platforms:

$$\text{scope}(P) = \sum_{p \in P} 2^{\,b(p)}$$

So a page on iOS and visionOS has scope 1 + 16 = 17, and `cupertino search "gesture" --only ios --only visionos` keeps the pages whose scope, with the Mac Catalyst bit masked off, equals 17:

$$\text{scope} \bmod 32 = 17$$

Three decisions in there are worth explaining, because each came from looking at the 1.5.0 database first:

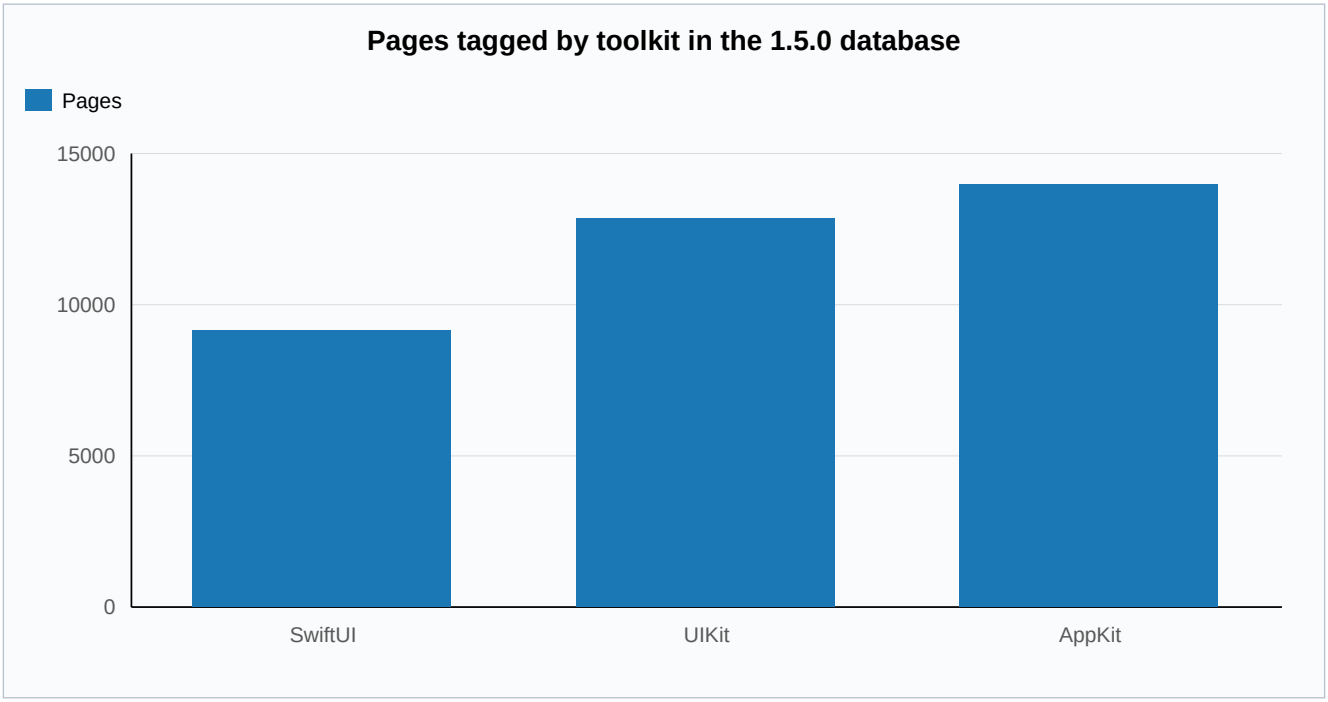
- **iPadOS is iOS.** It is never listed without iOS, so `ipados` is accepted as a spelling of `ios` and has no bit of its own.
- **Mac Catalyst is a marker, not a platform of the set.** It sits beside the set and does not break an exact match, so a page on iOS and Mac Catalyst still matches `--only ios`.
- **No scope is better than a wrong scope.** A list with an entry that is not one of the five platforms (`Unsupported OS`, Safari, DriverKit), an absent list, and a list of Mac Catalyst alone all store `NULL`, and a `NULL` never passes the filter, the same way an absent minimum never passes `--min-ios`.

The MCP `search` tool takes the same thing as `only`, a comma separated string.

## Filtering by UI toolkit: `--ui`

Issue #7 wanted the other axis: show me AppKit, not UIKit. `cupertino search "window" --ui appkit` now returns pages tagged AppKit. The names are `swiftui`, `uikit`, `appkit`, `catalyst` and `neutral`.

The tag is stored too, in `docs_metadata.ui_tags`, another bitmask (SwiftUI 1, UIKit 2, AppKit 4, Mac Catalyst 8). The framework decides the toolkit, whatever the source. In the 1.5.0 database that gives:




The interesting number is one that is not in that chart. If availability alone decided the tag, Mac Catalyst would tag **256,651 pages, 65 percent of the corpus**, and `--ui catalyst` would return Foundation. So Mac Catalyst is carried only together with a toolkit tag, and a page no rule tags is `NULL`, which the filter reads as toolkit-neutral. `--ui appkit` does not return neutral pages; `--ui neutral` selects them.

## A migration that silently did nothing

Adding a column to a database that ships to everyone is where this feature nearly shipped broken. The 19 to 20 migration added `platform_scope` and `ui_tags` with `NULL` in every row, which is correct. But `cupertino save` skips a page whose content hash and metadata stamp are unchanged, and those rows' stamps were unchanged. On a clone of the 1.5.0 documentation, a full save took 15 minutes, reported `indexed: 390925, skipped: 41`, and left **0 of 390,925 rows** with a scope or tags. `cupertino doctor` reported nothing wrong.

The migration now clears every stamp in the same transaction that adds the column. A missing stamp never equals the current one, so the next save rewrites every row. And until it has, `cupertino doctor` says so: `N of N rows not embellished (no metadata stamp)`, exit status 1, with the command that fixes it printed.

## `--swift` now means the same thing everywhere

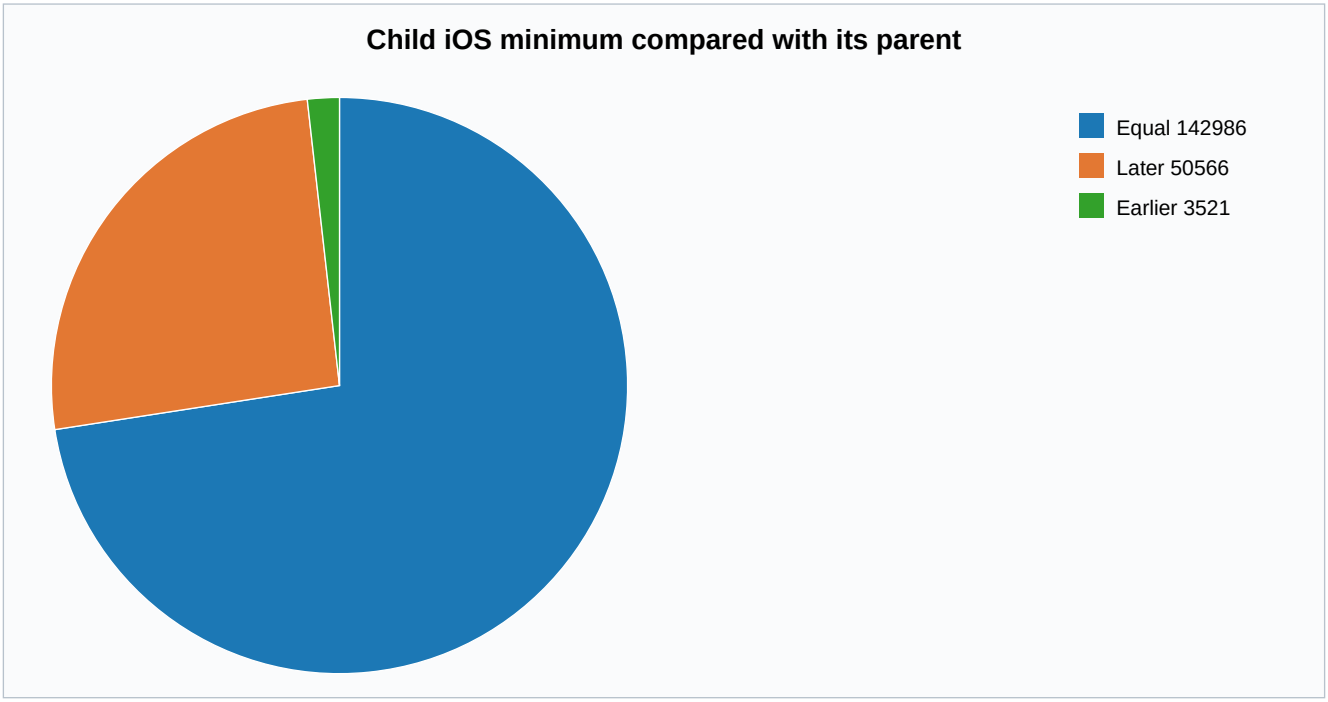
`--swift 5.9` means "usable with a toolchain of version 5.9". For packages that is `swift-tools-version` at most 5.9. For sample code it is the declared language mode or tools version. In both cases a project that declares nothing never passes, the same rule as the documentation. It works in the all-sources search, in the CLI and in MCP, which used to leave packages and samples out when the flag was set, or list them unfiltered.

The comparison is on an integer column, so the order is numeric, not alphabetical: 5.10 is above 5.9.

## The limits of version metadata, measured

I also wrote down what the version metadata cannot do, with numbers from the 1.5.0 database (#17):

- **70,113 pages** carry no minimum version on any of the five stored platforms: 61,841 have a list with no version in it, 8,272 have no list at all.
- Borrowing the parent page's version is not an option. Comparing the iOS minimum of a child with its parent, over 197,073 pairs:



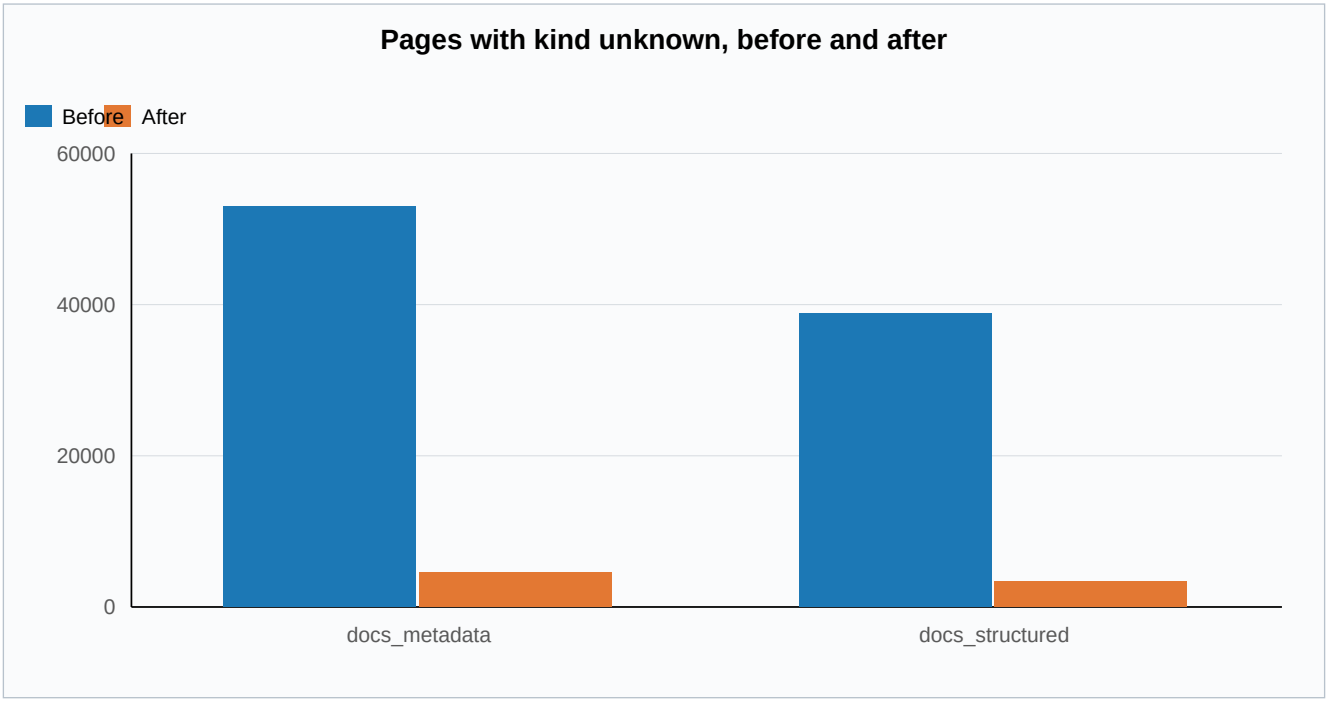

A child is later than its parent in 26 percent of the pairs and earlier in almost 2 percent. A guess from the parent would be wrong in 27 percent of the cases, so the filters guess nothing, and `--include-unknown` and the option help say so.

## Fewer "unknown" kinds

A page's kind (struct, method, property, article) drives ranking: a type ranks above its members. In 1.5.0, **52,941** pages had the kind `unknown`, and 38,886 in the structured table the ranker reads. Three fixes (#58) brought that down:

- Apple's role headings "Enumeration Case" (about 33,000 pages in the C frameworks: kernel, coreservices, iokit, accelerate), "Global Variable", "Instance Variable" and "Static Property" were not in the kind table. They are now.
- The web-service and device-management headings (Object, Dictionary, Web Service Endpoint, Property List Key) map to a kind.
- A page stored as `unknown` gets its kind again from the role heading in its own markdown, so a rebuild from the same files finally uses the improved tables.

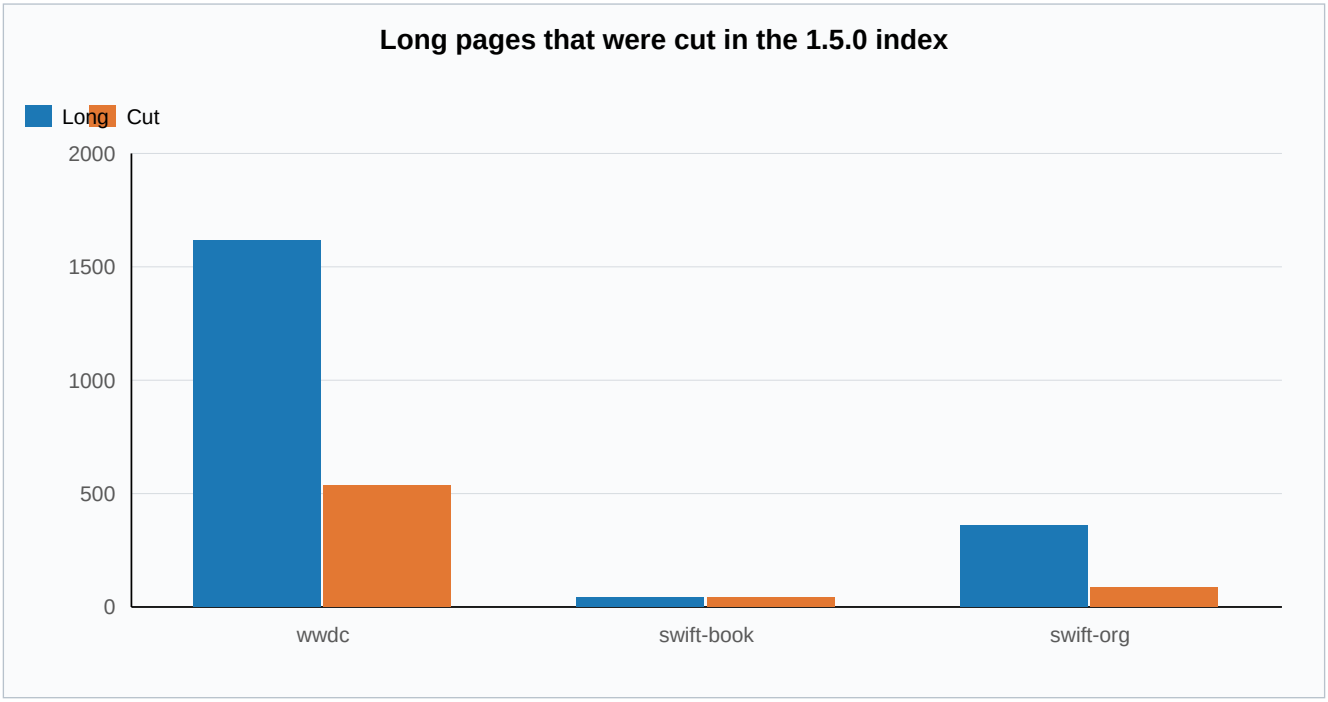
Rebuilding from the corpus files of 8 November:




A related fix (#57): an inline reference in Apple's structured data was written as `[doc://com.apple.appkit/documentation/AppKit/NSSearchToolbarItem]` where the name belongs. 59,360 pages of the 1.5.0 corpus held that form. The indexer now writes `[NSSearchToolbarItem]`, so the next save repairs them without a new crawl.

## Long pages were being cut

The full-text index keeps, for a page typed as a type or a member, its title, abstract, declaration and the first 2,000 characters of its overview. That is right for API reference and wrong for a WWDC session. For a page whose text names no kind, the kind was guessed from its first code block, so a session whose first example looked like `struct SomeInternalStruct` was indexed as a struct and the rest of its text could not be searched.




538 of 1,618 long WWDC pages. 40 of the 41 Swift book chapters: the chapter "Access Control" had **1,186 of its 34,150 characters** in the index. Sessions, notes, book chapters and swift.org pages are now indexed whole.

# Apple's tutorials

This is the new content, and the one that needed a parser.

Apple publishes tutorials on its web site, `developer.apple.com/tutorials/swiftui` and four more series, that are not in the documentation the database was built from. They are DocC JSON of a different shape. A documentation page keeps its text in `primaryContentSections`. A tutorial has none, so the existing parser would have kept a title and nothing else.

A tutorial keeps its text in `sections`: a hero (chapter, estimated time, Xcode requirement, introduction), tasks with numbered steps, and a quiz. Each step names a code file, a picture and a caption, and the files and pictures live in `references`, by identifier.



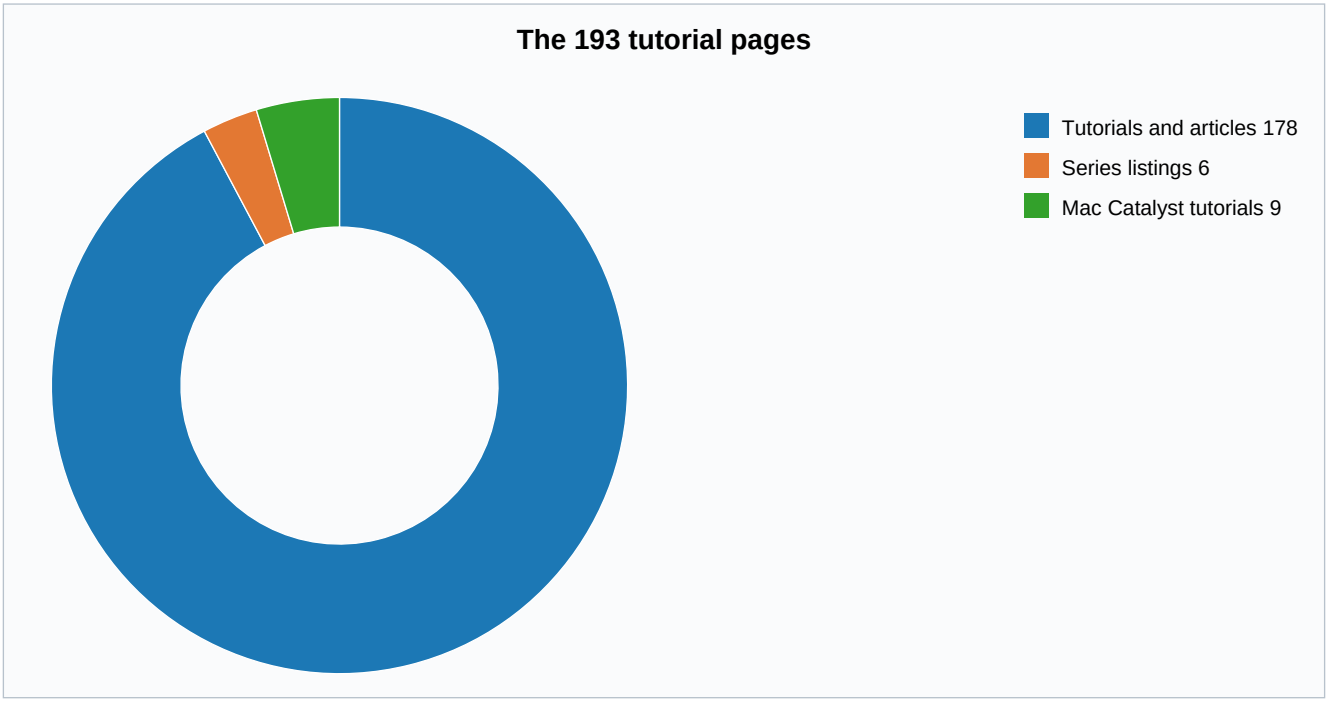

## How do you know the parser keeps everything?

A converter that "looks right" on three pages is not evidence. So the oracle was built apart from the converter: an independent walker reads the JSON and collects **every string a reader sees**, each text and code run, heading, title, code line of a step's file, quiz choice, justification and picture description. Every one of them must appear in the markdown. A control test shows the check can fail.

Run on all 178 tutorial and article pages Apple publishes: **178 complete, 0 refused.**

## The crawl

Five series are start pages for the crawl: `swiftui`, `swiftui-concepts`, `app-dev-training`, `develop-in-swift` and `sample-apps`. Each is crawled three links deep. Depth two missed the Mac Catalyst tutorials that App Dev Training holds as a sixth series, which is how depth became a setting of the start page.




Two things went wrong on the way, both worth the telling. The crawler follows links in the rendered page, and a tutorial's hero offered a project download, a zip, so the crawl tried to fetch zips as pages and logged errors. Fix: the hero names the file and does not link it, and the crawler rejects `downloads/`, `data/` and `assets/` paths. And when the pages were done, the corpus merge **held all 193 of them**: its safety rule says a page is well formed when its address names a documentation page, and a tutorial's does not. I taught the rule about tutorials, not the other way round, and a dry run then added 193 pages with none held.

A tutorial is addressed as `apple-docs://tutorials/<series>/<page>`:

```
$ cupertino read apple-docs://tutorials/swiftui/creating-and-combining-views
---
source:
https://developer.apple.com/tutorials/swiftui/creating-and-combining-views
---

# Creating and combining views

**Chapter:** SwiftUI essentials
```

## Proving I did not change the old pages

The rebuilt documentation database has 391,249 documents: the 391,056 of 1.6.0 and the 193 tutorials. I compared every old row with its 1.6.0 row, by content hash, kind, framework, language, the minimum and maximum of every platform, scope, UI tags, Swift version and word count: **0 differ, 0 missing.** The 16 WWDC sessions Apple lists that the database lacked (five Meet with Apple sessions and eleven older ones) went in the same way: 1,915 to 1,931 sessions, every existing row unchanged.

## Searches must not write

This one is the least visible and the one I like most.

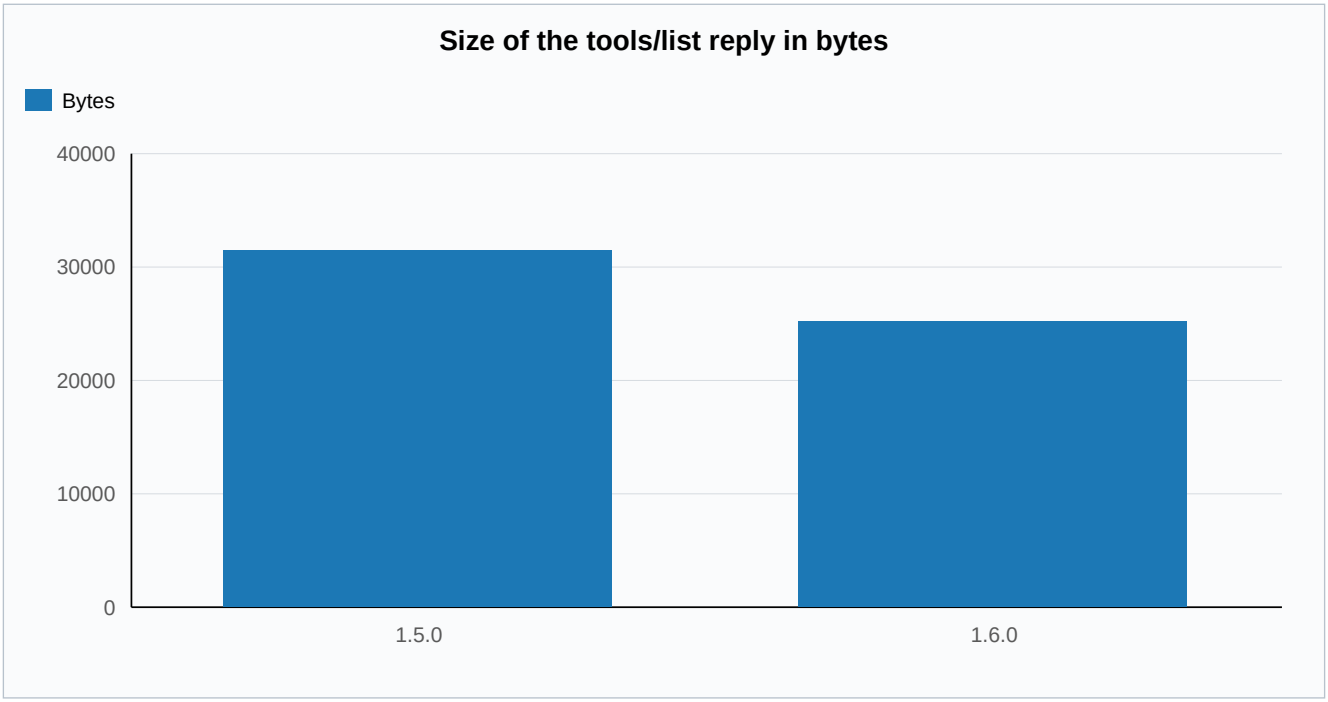
The search of all sources, the symbol searches and `get-inheritance` opened each database through the initializer that writes, and that one sets `PRAGMA journal_mode = WAL`. The databases ship in rollback mode. So the first all-sources search switched the shipped files to WAL: bytes 18 and 19 of the header changed from 1 to 2, the change counter went up, and a `-wal` and a `-shm` file appeared beside the database. Your installed files no longer matched the download, and a server killed with its log open left a log behind. That is what broke `cupertino setup` over the old databases (#115): setup renamed `x.db` and extracted a new one, but left the old `x.db-wal`, and SQLite replayed that log onto the new file.

The read paths now open read-only (#117). A database with an older schema is still migrated in place, as documented; a newer schema and a file with no schema are refused and left alone. Checked with the real binary: after an all-sources search, `search-symbols` and `get-inheritance`, the bundled files have the same SHA-256 and no sidecar exists.

The same family: `cupertino doctor` opens the search database read-only (#47), the release checkpoint reads the first page before asking SQLite to fold its log (#46, because a fresh connection knows nothing of WAL mode), and a second `cupertino save` no longer doubles the rows of a source (#45).

## Claude Desktop "stopped running"

A user reported (#93) that Claude Desktop showed "cupertino stopped running" and the first `search` call failed. My first reading was that the client deferred a large tool schema, so I shrank `tools/list` from 31,473 to 25,220 bytes, and I said in the changelog that this was a cleanup and not the fix.



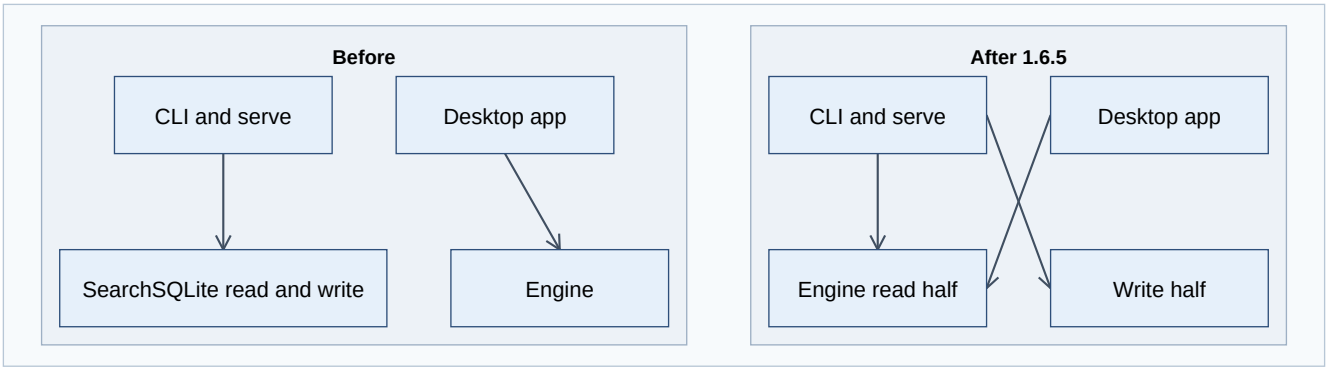

The real cause was my own reaper. Claude Desktop starts two `cupertino serve` processes within a second, one per client. The startup reaper killed every other `serve` of the same binary, so the second killed the first while its host was connected; the log shows the reap line one millisecond before the disconnect. A server is now stale only when its host is gone, meaning the kernel

re-parented it to launchd (parent PID 1). I also removed the 2,570-byte embedded icon from the `initialize` reply, which is now 218 bytes, and I say plainly that this is not claimed to be a cause.

# One engine instead of two (#122)

Cupertino has two implementations of "read these databases and rank the results": the one in this repository, used by the command line and `serve`, and `CupertinoDataEngine`, a separate package that the desktop app uses. Two implementations drift. When I checked, the engine could not open the 1.6.0 databases at all, because its schema numbers were stale (18, 4 and 5 against 20, 6 and 6), and no test in this repository noticed because nothing here called it.

The plan, written down before the code (`docs/design/read-through-data-engine.md`), is to make the engine the one read implementation and switch every caller to it. 1.6.5 carries the first half.




The read half of the search, the fused all-sources search and the read half of the sample index moved into the engine as they were, 62 files for the search alone, with only imports changed. The writers stay in cupertino, because they need SwiftSyntax and the engine must not.

## The oracle came first

Moving code is only safe if you can say what "the same" means. The answer key is the released binary. The comparison script runs the 1.6.0 binary and the candidate against one copy of the same databases, over every query set, in every filter configuration, on all nine databases and on the all-sources search, and compares the printed order and the rank or score of every result **to the last digit**.

Two things the first comparison did not see:

1. The engine's ranks were all 1.4617 times the binary's. The order was identical, so an order-only comparison passed. The cause was a different source lookup. The parity tests now compare the rank too, because a factor that scales every rank of a source is invisible to an order-only check.
2. Two results with the same score came out in a different order on each run, because the fused candidates were sorted by score alone, from the values of a dictionary. Equal scores are now ordered by source, then by identifier.

Result: 2,310 comparisons between 1.6.0 and a build with the shared code, 0 differ, 0 failed baseline runs, and the control passed. Samples: 28 of 28 queries equal, with and without a platform floor; the first run gave 14 and 12.

The engine used to fuse the per-source lists with a reciprocal rank fusion of its own:

$$\text{score}(d) = \sum_{i} \frac{1}{k + \text{rank}_i(d)}$$

It now builds the same candidate fetchers the binary builds and runs the binary's `SmartQuery` with the shipped source weights, so the two cannot differ.

One more detail about "to the last digit". The ranker applies each boost factor as its own `boost *= factor`, in a fixed order, because floating-point multiplication is not associative:

$$(a \cdot b) \cdot c \neq a \cdot (b \cdot c) \qquad \text{in floating point}$$

so when I split a 750-line search function into small ones, each factor kept its place in the product.

## Is it slower?

I measured it. `scripts/eval/SearchTiming.swift` runs the baseline and the candidate alternately, five runs each after a warm-up, over 48 cases, and prints the median of each, the baseline's own run-to-run spread, and the bound, median plus spread:

$$b = \tilde{t} + s$$

A candidate is a slowdown only if its median exceeds the baseline's bound. The control times the baseline against itself, which gives the false-alarm rate of the bound, about 4 percent. First result: sum of medians **+0.65 percent** (control -0.15 percent), no slowdown shown. That is a first measurement on 48 cases, not a proof.

## What is not done

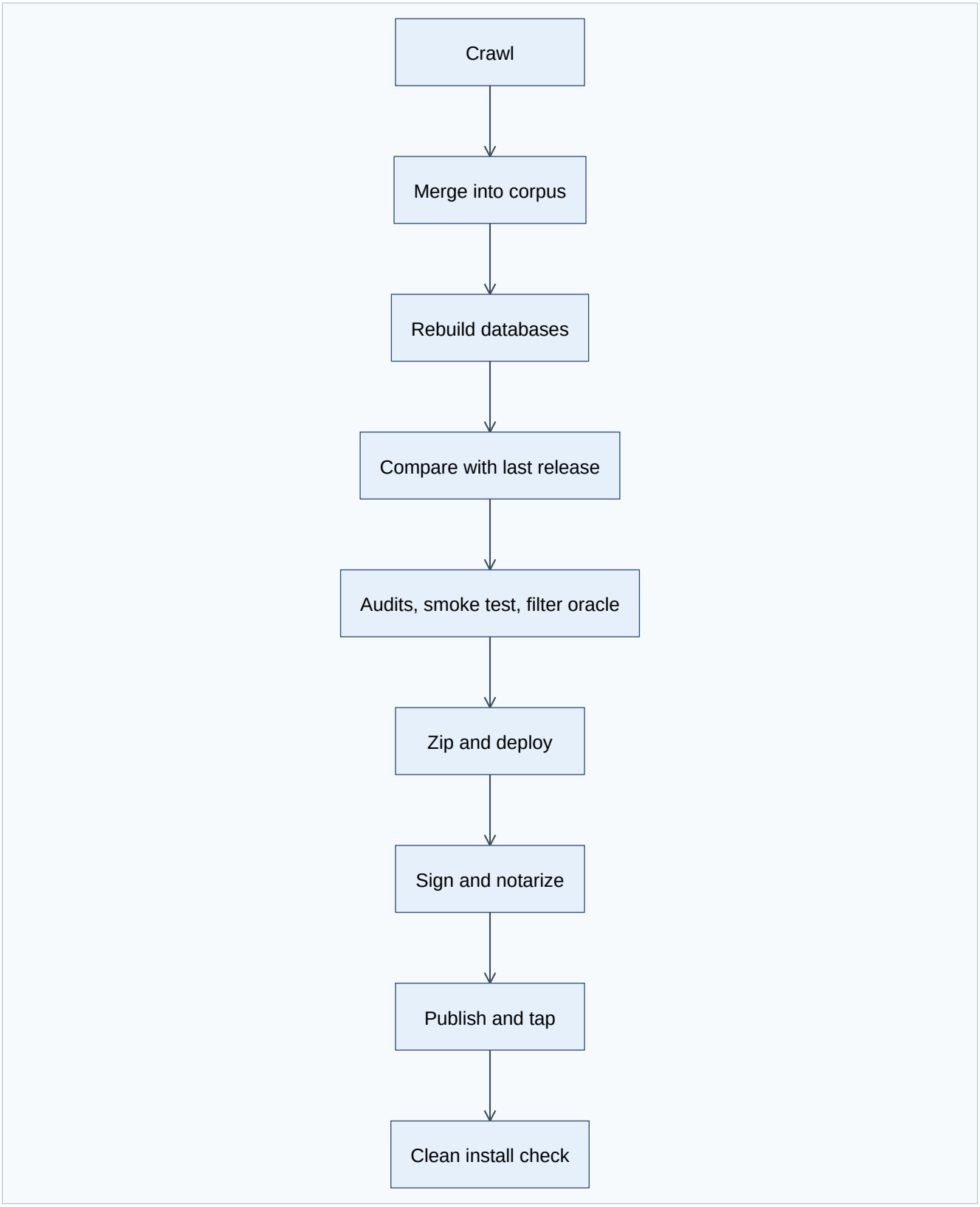
The command line, `serve`, MCP and the desktop app still read through the old implementation in the shipped binary, with the engine's code underneath for the parts that moved. Switching the callers, a structural gate that fails the build if one reads around the engine, and a repeat of the speed measurement on a fully configured candidate are open in #122.

## A small thing, quietly

Part of why I care about the engine: there is a desktop app. It is a native client for browsing the same documentation, Swift Evolution and sample code, on the Mac in SwiftUI and AppKit, and on iPhone and iPad in SwiftUI and UIKit. On the Mac it talks to `cupertino serve`; on iPhone and iPad it reads the databases itself, through the engine. It is early, it is not released, and I have no date for it. I mention it only because it is the reason the two read paths must not drift apart, and because the code is public if you want to look: cupertino-desktop.

# How a release gets built

Both releases went through the same pipeline, and writing down what went wrong is half the value.




- **The bundle step failed its own smoke test**, correctly. The copies of the seven unchanged databases had been switched to WAL by my own test runs, the exact behaviour of #117, and a read-only open of a WAL file with no `-shm` fails. I converted them back to rollback mode and checked bytes 18 and 19 of every header before the real run.
- **The verification took hours.** The filter oracle replays searches against the stored version metadata, more than 14,000 of them in this run, and passed.
- **A check was skipped, and I say so.** The "values against the raw corpus" oracles skip themselves when the build folder holds no corpus. For 1.6.5 they were skipped, and the

row-by-row comparison against 1.6.0 stood in for them.
- **The binary is signed and notarized**, built on the one Mac that has the signing identity, and I checked the published tarball by its checksum, its signature, the notarization assessment, and a clean-folder `cupertino setup` that then found a tutorial by search and read it.

## Upgrade

```
brew update
brew upgrade cupertino
cupertino setup
```

`cupertino setup` downloads the bundle for your version, 838 MB, and moves your old databases aside. Then:

```
cupertino search "creating and combining views" --source apple-docs
cupertino search "gesture" --only ios --only visionos
cupertino search "window" --ui appkit
cupertino search "actor" --source packages --swift 5.9
```

What is next: switching every caller to the engine, the gate that keeps it that way, and the open tracker items: #106 (a cut of 8,651 of 16,325 long Apple pages that needs a proper ranking evaluation)and #107 (a swift.org page that is a "Forbidden" error). The full list of changes, every one with its issue number and what was measured, is in the changelog.