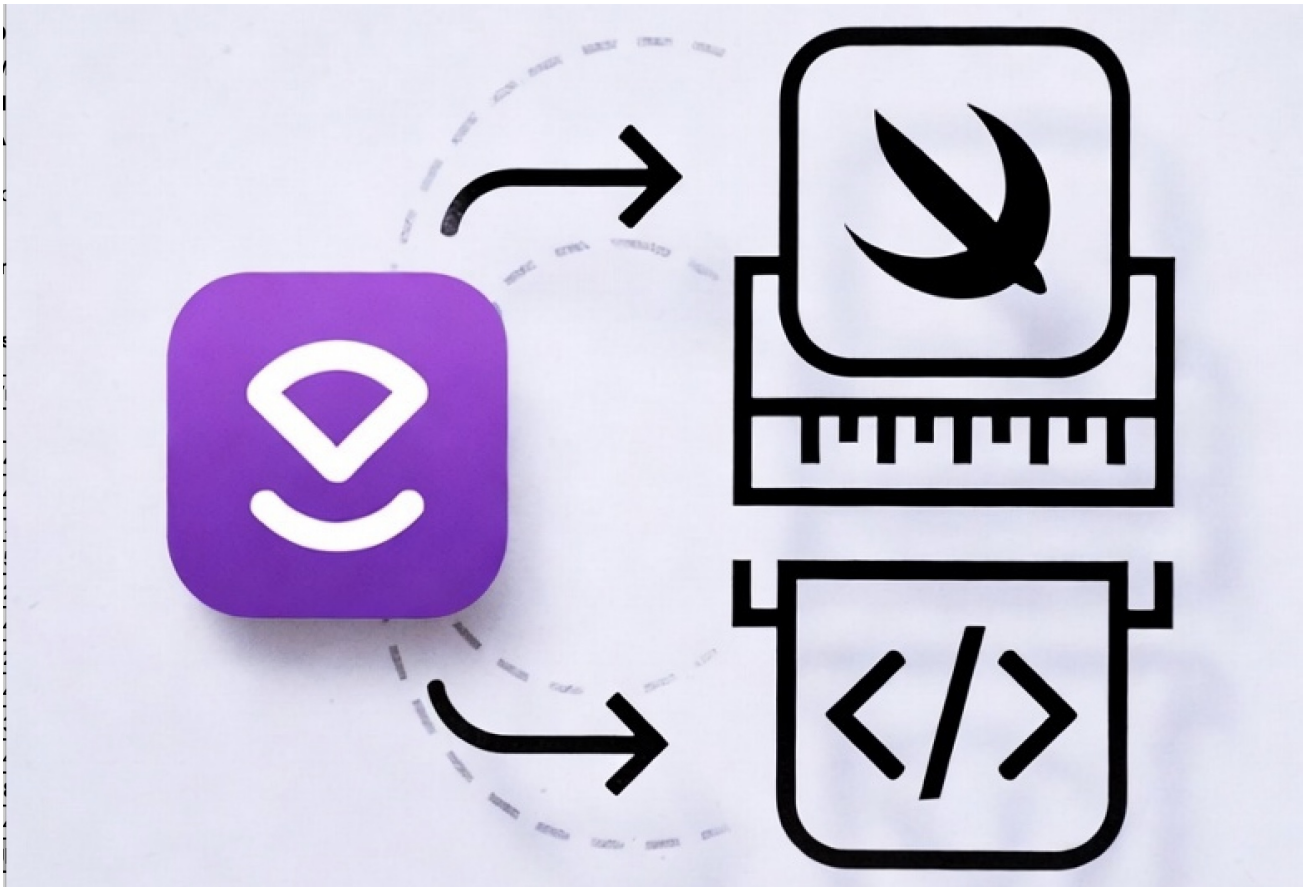

Cupertino v0.4.0: HIG Support, Framework Aliases, and Release Engineering Lessons

TL;DR: Human Interface Guidelines support, smarter framework search, and a hard lesson about release engineering.

Why jump from 0.3.4 to 0.4.0? Five issues closed: HIG support (#69), framework aliases (#91, #92), and Swift.org fixes (#93, #94). Enough new functionality to warrant a minor version bump.

What's New

Human Interface Guidelines Support

Cupertino can now crawl and index Apple's Human Interface Guidelines:

```
cupertino fetch --type hig
```

There's also a new `search_hig` MCP tool that lets AI agents search design guidelines with platform and category filters. When you ask Claude "what are Apple's recommendations for buttons?", it can actually look it up instead of guessing from training data.

Everything available via MCP tools is also available via command line:

```
cupertino search "buttons" --source hig
```

This is something people might not realize: you can query Cupertino the same way AI does. Every MCP tool has a CLI equivalent. Want to see exactly what Claude sees when it searches? Run the same query yourself. Useful for testing, debugging, or when you just want quick answers without a conversation.

Framework Aliases

Before, searching for "Core Animation" might not find results indexed under "QuartzCore" (the actual import name). Now there are 249 framework aliases in the database:

> `QuartzCore` ? `CoreAnimation` ? `Core Animation`
> `CoreGraphics` ? `Quartz2D` ? `Quartz 2D`

Search works regardless of which name variant you use.

Swift.org Fixes

The Swift.org crawler was broken. The base URL had changed from `docs.swift.org` to `www.swift.org/documentation/`, and the indexer was looking for `.md` files when the crawler was saving `.json` files. Fixed.

Why This Matters

Here's a real example from today. I asked Claude about Apple's Foundation Models framework - the new on-device ML APIs. Claude's response:

> "The MCP server pulls the complete Foundation Models sample code documentation - straight from Apple's current docs. No hallucinated API names!"

More examples of documentation you can search:

> **FoundationModels** - Apple's new on-device ML framework
> **Writing Tools** - System-wide AI writing assistance
> **Genmoji** - Custom emoji generation
> **Image Playground** - On-device image generation

This is also why Cupertino needs a human touch - it can't be fully automated. Someone has to notice when Apple adds new frameworks, when URLs change (like Swift.org did), when documentation structures shift. The crawler is automated, but the awareness isn't.

That's the whole point: deterministic, up-to-date docs instead of training data guesses.

Token Efficiency

A side benefit: Cupertino is more token-efficient than you'd expect.

Yes, tool results use tokens - documentation goes into context. But:

> **Accurate = fewer turns** - No hallucination means no correction loop
> **Curated content** - Returns exactly what's needed, not web search noise
> **Local search is free** - No API cost for the search operation itself
> **Truncated summaries** - Results are summarized; full doc only when you call `read_document`

So not zero tokens, but more *efficient* token use. And no external API costs for search.

The Archive Discovery Problem

Here's something interesting I discovered while testing. When Claude searched for "CALayer animation", it got API reference pages - classes, methods, properties. Claude wanted to dig deeper:

> "Let me try a more targeted search - something like the layer tree architecture or model/presentation layers that would be covered in the Core Animation Programming Guide."

Then I suggested searching with `source: "apple-archive"`. Claude's reaction:

> "Now THAT's the difference! When searching with source: apple-archive, we get the Core Animation Programming Guide and the Animation Types and Timing Programming Guide - the real meat."

The conceptual deep-dives that explain *why* things work the way they do.

Modern Apple docs = API reference (the *what*) Archive docs = Conceptual guides (the *why*)

The problem? AI agents don't know to search the archive. It's an opt-in parameter buried in the tool description.

The Solution: Auto-Surface Archive Teasers

I'm planning to modify search results to automatically hint at archive content:

```
# Search Results for "CALayer animation"

## Modern Documentation
[1] CALayer - API reference...
[2] CABasicAnimation - API reference...

---

## ? Related Archive Guides

**Core Animation Programming Guide** - Layer geometry, animation timing...

> For full archive content, search with `source: "apple-archive"`
```

This way:

- Modern docs still come first
- AI agents always see archive exists
- They learn how to get more, in context

Only show the archive section if there are relevant results. Same principle could apply to sample code, Swift packages, or any other source.

The Release Process Disaster

Here's where it gets interesting. I had all the code ready, all the tests passing, and I was ready to ship.

The process seemed simple:

1. Bump version in `Constants.swift`
2. Update `README.md` and `CHANGELOG.md`
3. Create git tag
4. Push tag (triggers GitHub Actions build)
5. Upload databases to `cupertino-docs`
6. Update Homebrew formula

What could go wrong?

The Tag Timing Problem

I merged my feature branch, created the tag, pushed it... and then realized I hadn't committed the version bump to `main` yet. The tag pointed to a commit where `Constants.swift` still said `0.3.5`.

GitHub Actions dutifully built a beautiful, signed, notarized universal binary... that reported version `0.3.5`.

```
$ cupertino --version
0.3.5
```

When users ran `cupertino setup`, it tried to download databases from `v0.3.5` instead of `v0.4.0`. Everything was broken.

The Fix

I had to:

1. Delete the tag on GitHub
2. Delete the local tag
3. Make sure the version bump commit was pushed to `main`
4. Verify the built binary reports correct version **before** tagging
5. Recreate the tag
6. Wait for GitHub Actions to rebuild
7. Update the Homebrew formula with the new SHA256

The SHA256 Dance

When I first updated the Homebrew formula, I grabbed the SHA256 from the old (broken) binary. After rebuilding, the checksum changed. Users got:

```
Error: Formula reports different checksum: cf035352...
SHA-256 checksum of downloaded file: 5c5cf7ab...
```

Another round of updating the tap.

The 11-Step Release Process

After today's adventures, here's what the release process actually looks like:

1. Update version in `Constants.swift`, `README.md`, `CHANGELOG.md`
2. Commit and push to main
3. Build locally and verify `--version` matches
4. Create and push tag
5. Wait for GitHub Actions (~5 min)
6. Create GitHub release with notes
7. Build locally and install
8. Upload databases with `cupertino release`
9. Get new SHA256 from release
10. Update Homebrew tap formula
11. Verify on fresh machine

That's 11 steps across 4 repositories. Miss one, and you're rebuilding everything.

Time for Automation?

I'm seriously considering writing a release script. Swift or Bash?

The pragmatic choice is Bash - it's just orchestrating CLI commands. But I'm a Swift purist. The `cupertino release` command already handles database uploads with proper GitHub API integration. Extending it to handle the full workflow feels right.

Something like `cupertino release --full` that:

1. Checks for uncommitted changes
2. Verifies version consistency
3. Builds and validates `--version`
4. Creates and pushes the tag
5. Waits for GitHub Actions
6. Uploads databases
7. Updates the Homebrew tap

One command. No mistakes. Written in Swift.

Lessons Learned

1. **Order matters.** Commit the version bump before creating the tag.
2. **Verify before you ship.** Build locally and check `--version` before tagging.
3. **Document your release process.** I now have a detailed `DEPLOYMENT.md` with warnings.
4. **Automate what hurts.** If you make the same mistake twice, write a script.

What's Next

Enhanced search results - Auto-surface archive, samples, packages in every search (high priority)

Release automation - `cupertino release --full` in Swift

Fix setup animations - They're broken

Keep crawling - Fresh docs matter

This release taught me a lot. Not just about release engineering, but about how AI agents actually use documentation. The archive discovery problem was a surprise - valuable content hidden behind an opt-in flag that agents don't know to use.

Building tools for AI is different. You're not just building for humans who read documentation. You're building for agents that learn from tool descriptions and in-context hints.

Key discovery: Claude reads tool descriptions once, but reads search results every time. If you want agents to use a feature, put it in the output - not buried in documentation they'll forget. Every search result is a teaching moment.

Next up: implementing enhanced search results. Once that ships, every search will auto-surface relevant content from archives, samples, and packages - with the exact commands to dig deeper. No more hidden treasure.

Cupertino is an Apple Documentation MCP Server. Install with `brew install mihaelamj/tap/cupertino`*. Check it out at github.com/mihaelamj/cupertino.*