Apple TestFlight Debug Bible ## Free 5-Page Preview --- > **"Could not install \[App Name\]. The requested app is not available or doesn't exist."** If you're reading this, you've almost certainly seen that error. You've checked everything you can think of. Your build is `VALID`. Your tester is `ACCEPTED`. The green dot is there in App Store Connect. And TestFlight still refuses to install. You are not doing anything wrong. You are hitting a 4-year-old Apple infrastructure bug that has zero confirmed fixes on Apple's own developer forums. --- ### What this preview covers This free preview gives you the real substance — not a teaser. By page 5 you will: 1. **Understand exactly what the bug is** and why Apple hasn't fixed it 2. **See the 8-class root-cause taxonomy** used to rule out every known non-bug cause 3. **Run the first diagnostic step** yourself with a real script output example 4. **Understand the workaround** and *why* it works at the cache layer 5. **Know what to do next** — whether that's the free workaround or the full 62-page reference with all scripts --- *By Sun — Tokyo, May 2026 · jiejuefuyou.github.io* *Full edition: 62 pages · 3 Python scripts (765 lines) · 8-class taxonomy deep dives · Forum thread index (4 years of data) · ASC API endpoint cheatsheet · UI-only walkthrough*

Page 2: The Bug — What It Is and Why It's Still Alive ### The 4-year timeline The canonical Apple Developer Forum thread for this bug is **#674932**, created February 2021. As of May 2026 — **more than four years later** — it has: - 400+ replies - Zero confirmed Apple fixes - Zero Apple staff answers confirming a root cause A newer thread, **#809636**, appeared December 2025. At publication: zero replies from Apple. That's not a gap in Apple's forum monitoring. That's the bug. ### What the bug actually looks like The error fires *after* tapping Install in the TestFlight app. Before that moment, everything looks correct: - The app appears in TestFlight with the right version number - App Store Connect shows the build as "Testing" with a green dot - Your tester account is in the correct Internal Testing group with `state=ACCEPTED` - The build's `processingState=VALID` per the ASC API - No email from Apple indicating any processing failure Then you tap Install. "Could not install. The requested app is not available or doesn't exist." It fires the same way on every device you try. It's not network-related. It's not corrupted iOS. It correlates with one thing only: **whether the app has ever reached `READY_FOR_SALE` in its lifetime**. Apps that have been on the App Store before — even once — don't hit this. Apps that have never crossed into App Store distribution hit it reliably. ### The 8-class root-cause taxonomy Before concluding you're hitting the 4-year bug, you must rule out 7 other causes. The diagnostic framework groups them into 8 classes: | Class | Name | Ruling it out | |---|---|---| | 1 | Tester state | ASC API: `betaTesters/{id}` → `state=ACCEPTED` | | 2 | Build state | ASC API: `builds/{id}` → `processingState=VALID`, `expired=false` | | 3 | Apple ID conflict | Manual: confirm tester's Apple ID matches invite email | | 4 | Paid Apps Agreement | ASC web UI: agreements status = ACTIVE | | 5 | Distribution cert + profile | Certs, IDs & Profiles: cert active, profile not expired | | 6 | iOS minimum version | ASC API: `builds/{id}` → `minOsVersion` ≤ device iOS | | 7 | Export compliance | ASC API: `builds/{id}` → `usesNonExemptEncryption` not null | | **8** | **Distribution cache misroute** | **Not checkable via API — only confirmed when 1-7 all PASS** | Class 8 is the actual 4-year bug. Classes 1-7 are the 30-day spiral most developers get trapped in. The diagnostic script in the full edition (`asc_diag.py`) runs classes 1-7 automatically via JWT-authenticated API calls across all your apps in a single command. Page 3 shows you real output.

Page 3: First Diagnostic Step — What You Run and What You See ### Run asc_diag.py --all The diagnostic script is included in full source in the complete edition. The command you run:

```
export ASC_KEY_ID=XXXXXXXXXX
export ASC_ISSUER_ID=xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
export ASC_KEY_FILE=./AuthKey_XXXXXXXXXX.p8

python orchestrator/asc-tools/asc_diag.py --all your-tester@icloud.com
```

This runs all 7 ruleable classes against every app in your Apple Developer account. It takes under 90 seconds for a 4-app portfolio. ### Real output — AutoChoice (2026-05-07)

```
═══ com.jiejuefuyou.autochoice (6765667062) ═══
  found build: 1.0.0 (id=4a7b9c12-...)
  found internal group: 8e4d2a7f-...
  [PASS] Tester state: ACCEPTED
  [PASS] Build state: VALID, expires 2026-08-04
  [WARN] PAID_APPS_AGREEMENT_UNSIGNED: verify manually at ASC web UI
  [PASS] Distribution cert: active until 2027-01-15
  [PASS] minOsVersion: 16.0
  [PASS] Export compliance: false (declared)
  [INFO] Class 8 candidate: app has never reached READY_FOR_SALE
```

That `WARN` on Paid Apps Agreement led to a manual check — it was actually signed correctly, the API just can't confirm it (Apple doesn't expose that endpoint). The important line is at the bottom: `Class 8 candidate: app has never reached READY_FOR_SALE`. Six PASS results plus the Class 8 candidate flag is the definitive confirmation you're hitting the 4-year bug, not a configuration mistake. ### Decision tree: first three levels

```
TestFlight install fails with "not available or doesn't exist"
│
├── Is tester state = ACCEPTED?
│   └── No → Fix: use betaGroups/{id}/relationships/betaTesters PATCH
│            Caution: state mutations post-2024 require group endpoint
│
├── Is build processingState = VALID?
│   └── No → Fix: check upload email from Apple; wait up to 2h for PROCESSING
│            Caution: builds expire at 90 days — check expiry first
│
├── Is Paid Apps Agreement ACTIVE?
│   └── No → Fix: sign at appstoreconnect.apple.com → Agreements, Tax, Banking
│            Caution: agreement can de-activate after banking changes
│
└── Do all 7 classes PASS?
```

```
    ├── No → Work the failing class (see full taxonomy, Part 2)
    └── Yes → You are hitting Class 8. Go to Page 4.
```

The full decision tree in the complete edition extends 8 levels deep and covers edge cases including: tester records per-app scope, build cert chain mismatch, export compliance null vs false distinction, Apple ID conflict detection even when the tester "looks" accepted, and the iOS 18 deployment target floor issue.

Page 4: The Workaround — Submit for Review (Manual Release) ### What you actually do When all 7 classes pass and the app has never reached `READY_FOR_SALE`, the fix is: **Submit for App Store review with `releaseType=MANUAL`.** Manual Release means: if Apple approves the build, the app sits in `PENDING_DEVELOPER_RELEASE` state indefinitely. You control when (and whether) it goes public. You can wait months. You can hold it forever. The App Store listing only goes live when you explicitly click "Release." The workaround does not require you to publish your app publicly. It requires you to let the build cross into Apple's review pipeline, which triggers the metadata cache rebuild that TestFlight needs. ### Why this works The hypothesis (supported by 4 years of forum data): TestFlight's install pathway depends on a **metadata cache layer** that only fully populates when an app enters the App Store distribution pipeline. In Apple's infrastructure — which has been progressively virtualized into regional CDN caches since 2020-2021 — a new app's metadata remains in an "incomplete" state until the review pipeline touches it. The cache rebuild is a side effect of the submission process, not of public release. Once the cache is populated: - TestFlight install works immediately - The fix persists across future builds - You don't need to re-submit to maintain it This is confirmed by the forum pattern: roughly 25% of reporters in thread 674932 said the bug "magically resolved" after their app got approved for the App Store. None of them understood why. The cache rebuild explains it. ### Pre-flight checklist before submitting Before you run the submit script, you need: - [ ] At least one screenshot set uploaded per required locale (Apple won't accept submission without screenshots) - [ ] App description filled in (minimum 10 characters — a placeholder works) - [ ] Age rating declared - [ ] Export compliance declared (`usesNonExemptEncryption=true/false`) - [ ] Build attached to the App Store version - [ ] `releaseType=MANUAL` — **critical**. Without this, Apple approval means immediate public release. The full edition includes `asc_submit_for_review.py` (full source, 280 lines), which handles the pre-flight checklist programmatically, sets `releaseType=MANUAL` automatically, and uses the current `reviewSubmissions` API (the deprecated `appStoreVersionSubmissions` endpoint silently fails — the full edition covers this). ### After submission Once submitted, you enter Apple's review queue: typically 24-72 hours. The full edition includes `asc_submission_monitor.py` — a state machine watcher that polls Apple's API on a configurable interval and sends a notification when the build moves from `WAITING_FOR_REVIEW` → `IN_REVIEW` → `PENDING_DEVELOPER_RELEASE`. **Confirmed replication** (2026-05-07): DaysUntil (one of my 4 apps) successfully unblocked via this workaround. Submission at 11:58 JST. State: `WAITING_FOR_REVIEW → IN_REVIEW`. TestFlight install started working before approval completed. That confirms the cache rebuild fires at submission entry, not at approval.

Page 5: What You Have After Reading This ### The 5 things you now know 1. **The bug is real, not you.** Apple forum thread 674932 has been open since February 2021 with 400+ replies and zero Apple fixes. You are not making configuration mistakes. 2. **There are 7 other causes.** Classes 1-7 of the taxonomy cover the non-bug root causes. If any of them are failing, the fix is local and fast. The diagnostic script runs all 7 in 90 seconds. 3. **You can confirm Class 8 with data.** When all 7 pass and the app has never reached `READY_FOR_SALE`, you have the evidence. The script prints `Class 8 candidate: app has never reached READY_FOR_SALE`. That's your diagnosis. 4. **The workaround is free and reversible.** Submit for review with `releaseType=MANUAL`. Apple approves. TestFlight cache rebuilds. You hold the app in `PENDING_DEVELOPER_RELEASE`. You never have to go public if you don't want to. 5. **The cache rebuild fires at submission, not approval.** Confirmed on DaysUntil 2026-05-07. TF install worked before the review completed. You don't have to wait 72 hours to see the fix take effect. --- ### What the full 62-page edition adds What you just read is the diagnostic and workaround path. The full edition adds everything needed to execute it without debugging the execution: | What | Full edition | This preview | |---|---|---| | 8-class taxonomy (overview) | Yes | Yes | | 8-class taxonomy (deep dives, edge cases) | Yes | No | | asc_diag.py full source (255 lines) | Yes | Output sample only | | asc_submit_for_review.py full source (280 lines) | Yes | No | | asc_submission_monitor.py full source (230 lines) | Yes | No | | Decision tree levels 4-8 | Yes | Levels 1-3 only | | Forum thread index (4 years, 8 threads mapped) | Yes | Thread #674932 + #809636 | | ASC API endpoint cheatsheet | Yes | No | | UI-only walkthrough (no ASC API auth required) | Yes | No | | 14-day no-questions refund | Yes | — | The full edition is $29. That's less than 2 hours of any iOS consultant you'd otherwise hire for this problem — and consultants who know this bug still charge their hourly rate for the queue-waiting time. --- ### Get the full edition **[TF Debug Bible — $29 on Gumroad →](https://jiejuefuyou.gumroad.com/l/tf-debug-bible)** 14-day no-questions refund. If you read it, run the scripts, and the workaround doesn't unstick you, email jiejuefuyou@gmail.com. --- *Questions? Same email. I reply within 24 hours.* *jiejuefuyou.github.io — Tokyo, May 2026*