ChatGPT

Challenges in Retrieving App Names/Icons in iOS Screen Time Apps

Introduction

Building a parental control or screen time app on iOS that monitors a **child's device** and displays usage details (app names, icons, durations) on a **parent's device** is notoriously challenging. Apple's Screen Time frameworks (FamilyControls, DeviceActivity, etc.) were designed with **privacy in mind**, which often means third-party apps get only opaque data instead of clear app identifiers [1]. This makes it difficult to directly fetch the names or icons of apps used on the child's device for display elsewhere. Developers of similar apps have shared various workarounds and troubleshooting insights in forums and open-source projects. Below, we delve into these challenges and the solutions (or hacks) others have attempted.

Apple's Screen Time API and Privacy Limitations

Apple's official Screen Time APIs (FamilyControls, ManagedSettings, DeviceActivity) introduce a token-based system. When a child's device authorizes tracking of certain apps via the `FamilyActivityPicker`, the result is a `FamilyActivitySelection` containing **cryptographic tokens** for each selected app or category [1]. These `ApplicationToken` objects intentionally hide the app's identity. As developer Frederik Riedel notes, *"apps… don't get any information on which target apps the user actually selects… all information is hidden behind cryptographic tokens."* [1]. In practice, each app token is an opaque blob that **cannot be reverse-mapped** to an app name or bundle ID by the app.

Apple does provide a way to **display** the app's name and icon **without revealing it to the code**. For example, SwiftUI offers a `Label` initializer that takes an `ApplicationToken` and will render the real app name and icon in the UI [2]. However, the app's code still cannot inspect those values directly. On Apple's Developer Forums, engineers confirmed that you "aren't allowed/supposed to rehydrate the app selection tokens" into identifiers in the main app [3]. Any attempt to create an `Application` from the token and read its `bundleIdentifier` or `localizedDisplayName` will yield `nil` outside of the approved context (like certain extensions) [4] [5]. In short, **the main app cannot programmatically get the app's name or ID for a given token** – this is *"for privacy reasons"*, as one Stack Overflow expert put it [6].

The official approach is to use Apple's UI or extensions to show the information. For instance, one developer discovered that using `Label(token)` allowed them to at least **display** the selected app's name/icon to the user [7]. Likewise, Apple later introduced a `DeviceActivityReport` extension (iOS 17+) which runs in a separate process with elevated privileges to access usage data. Within such a **report extension**, the code can iterate over usage records and obtain each app's `localizedDisplayName` for display [8]. But again, this happens in an isolated context — the parent app itself still doesn't receive the raw names. The data is meant to stay sandboxed (e.g. the extension can render a chart or list of apps used, which the parent app can embed as a view, but not extract as plain text).

Developer Forum Insights and Troubleshooting

This limitation has been a hot topic on developer forums and Q&A sites. An Apple Developer Forums thread from April 2025 shows a developer asking *"How to get the bundleIdentifier or app name from FamilyActivitySelection's applicationTokens?"* after observing that `Application(token: token).bundleIdentifier` was always nil [4] [5]. The responses confirmed there is **no supported way to directly get those details** in the main app. In that thread, another developer noted that apps like *Opal* somehow show the app names, wondering if they found a workaround [3]. The consensus was that Apple does **not** expose a direct mapping; any solution would be non-trivial. One commenter even half-jokingly speculated that perhaps the app took a **screenshot** of Apple's picker UI (which displays the names/icons) and used image recognition to identify the apps [9] – a far-fetched idea illustrating developer frustration.

On Stack Overflow, similar questions have been asked. In one case, a developer noticed that the popular app **One Sec** manages to display the names of blocked apps and wondered how to achieve this in their own app. The accepted wisdom: *"You cannot retrieve that information for privacy reasons. You only get an opaque token… The bundle id and name is only available in a shield extension."* [6]. In other words, one must either use Apple's provided UI (Shield/Report extensions or Labels) or forgo the data entirely. The original asker eventually commented that they resorted to using `Label()` so the user could see the app name and icon in the interface [7], even though the app's logic doesn't possess the actual string/icon.

Another Reddit discussion confirms this: *"Apple does not provide an API so developers can know details about other apps. ScreenTime API gives you tokens which really don't mean anything outside of the user's device."* [10]. In that discussion, a user explains that **Opal** (a screen time app) likely had to get very creative: it reportedly uses a custom local VPN to monitor network traffic and correlate it with Screen Time tokens, effectively deducing which token corresponds to which app by network behavior [11]. This is an example of how far developers have gone to bypass the opaque token limitation.

Approaches Used by Similar Apps

Given these hurdles, developers of screen time and parental control apps have employed a range of strategies:

- **Relying on Apple's Framework (with UI Workarounds):** Some apps stick to the sanctioned Screen Time APIs and simply use the UI capabilities to show app names/icons. For example, *One Sec* and others use the FamilyControls pickers and shields. The parent app might not "know" the app names internally, but it can still present a list of "Top apps used" to the parent through Apple's `DeviceActivityReport` UI extension. The code running in that extension can fetch usage durations and app display names (as shown in Apple's documentation) to populate the view [8]. The downside is lack of flexibility — the data can't be easily customized or sent to a server, because doing so would violate Apple's rules (the API forbids exporting Screen Time data off the device [12]).

- **Mobile Device Management (MDM) Profiles:** Several earlier parental control solutions turned to MDM. By **supervising the child's device** (via Apple Configurator or DEP enrollment) and installing a special MDM profile, a parent app/service can query the list of installed apps (including each app's name, bundle ID, and icon if needed) and even enforce restrictions. In fact, an MDM command exists to retrieve the **InstalledApplicationList** on a supervised device, which includes app names and

identifiers. Apps like **Qustodio** leverage this approach: Qustodio's iOS product requires the parent to install a configuration profile on the child's phone [13], effectively enrolling it in an MDM. This allows Qustodio to know what apps are present and even prevent the child from deleting the Qustodio app or certain apps (via restrictions) [14] [15]. Historically, Apple was wary of consumer apps using MDM (citing guideline 2.5.1 about using APIs only for intended purposes) and even removed some parental control apps in 2019 for this misuse [16]. Apple later carved out an exception for parental control apps, but they **must strictly use MDM for parental monitoring purposes** and not, say, marketing data collection. In summary, the MDM route can yield app names and icons (since the service knows the bundle IDs and can fetch icons via the App Store API or iTunes lookup), but it requires a complex setup (enterprise dev account, user trust of a profile) and careful compliance with Apple's rules [16].

- **VPN and Network Traffic Analysis:** Some modern screen-time apps bypass Apple's Screen Time APIs entirely for usage tracking. A prime example is **Opal**, which explicitly states that *"iOS Screen Time is not available as a public API. We decided to build our own."* [17]. Opal runs a **local VPN** on the device that never routes data to an external server but instead monitors network connections of apps. By observing which domains an app contacts (and when), Opal can infer which app is active and for how long [18]. For instance, if it sees continuous requests to *.instagram.com*, it concludes Instagram is in use and can time it. This method also allows blocking Internet access to those apps on the fly (simulating an app block). Using a VPN for this purpose is clever because it operates within Apple's allowed frameworks (Network Extensions) and doesn't require special Screen Time entitlements – but it only tracks online activity. An offline app that doesn't hit the network might not register, so there are limitations. **Qustodio** also uses a VPN on iOS: the VPN is how it "helps you monitor your child's device" and block content [19]. In essence, these apps trade the rich data of Apple's Screen Time (which they can't fully access) for a more indirect measure of app usage via network calls. It's less precise in some cases, but it does give them the app's identity (because the VPN sees the traffic's destination and thus knows which app process is sending it).

- **Hybrid Approaches (Screen Time API + Extras):** Some apps combine methods. **Jomo**, for example, appears to use Apple's Screen Time API for certain features (requiring the Screen Time permission), but it also computes its own **"Estimated Screen Time"** metric to overcome gaps in Apple's data. According to Jomo's help center, Apple doesn't allow access to Screen Time data in some contexts like widgets, and sometimes the native data "isn't accurate enough" for their needs [20]. To compensate, Jomo uses a secondary method to calculate usage time. They don't reveal their secret sauce, but it could involve reading device logs, using the VPN trick, or other heuristics. The result is that Jomo shows two figures – one is Apple's official screen time (raw data), and one is their own estimate [21]. This indicates Jomo is likely using the Screen Time API to get the total usage for apps when possible, but if that fails or is delayed (Apple's data might only sync infrequently), they fall back on their own continuous tracking. Jomo even advises users to turn off "Share Across Devices" in Apple's Screen Time settings to make Apple's data more reliable for a single device [22] – highlighting the complexity of juggling multi-device Screen Time info.

- **Open-Source Projects and Developer Experiments:** There are also open-source libraries and discussions that shed light on these issues. The `react-native-device-activity` library, for instance, wraps Apple's Screen Time APIs for React Native apps. Its documentation notes "weird behaviors" and emphasizes that these APIs **require special approval from Apple** before an app can ship [23]. Many developers have shared frustration that Apple's documentation is sparse and the

frameworks are buggy or unintuitive [24]. Some have tried creative solutions like periodically reading system logs or using private APIs (e.g. `LSApplicationWorkspace`) to list apps, but those approaches are not App Store-safe (private API use will get an app rejected unless the device is supervised and the app has special entitlements). In supervised mode, one *could* use `MobileInstallation` or MDM channels to fetch app info, but again, that veers into MDM territory.

In summary, **commercial parental control apps often resort to out-of-band techniques** to get app names and icons. VPN-based monitoring is common (e.g. Opal, Qustodio) because it sidesteps Apple's token system entirely. MDM is used by services that need deeper control (device supervision, app list retrieval), though it comes with user friction (installing profiles) and oversight by Apple. Apps that stick within Apple's Screen Time framework must accept that they can only display app names/icons via Apple's provided UI components or extensions – they cannot freely collect and transmit that info themselves [12].

Notable Examples and Developer Solutions

- **Stack Overflow Case:** A developer (Liam) in 2025 asked how to get app names from the FamilyActivityPicker selection, noting *"I noticed that One Sec successfully manages to retrieve app names... Currently I only get nil."* The solution was that **you simply can't get those names in Swift code** – the data is sandboxed [6]. Instead, Liam discovered that using the `Label` with the token would at least show the user the correct name/icon in the UI [7]. This mirrors Apple's intended usage: let the system draw the sensitive info, without your app handling it directly.

- **Apple Dev Forums Case:** A thread on Apple's forums discussed how apps like **Jomo** seemingly show screen time stats for both the child and parent devices. The question wondered if there was a private API to get the parent's own usage data or the child's data on the parent's phone [25] [26]. No clear answer was given (likely because Apple provides no public API for a parent to pull a child's stats remotely, apart from the DeviceActivityReport which runs on the child's data). This implies that Jomo might be doing something creative, such as having the child's device upload usage stats to a cloud service (which Apple's Screen Time API technically forbids – it's meant to be on-device only [12]), or using an approved MDM channel. The lack of an official solution here is telling.

- **Opal's Public Statements:** Opal's engineering team openly shared that *"Today, iOS Screen Time is not available as a public API. We decided to build our own."* [17]. They confirm that no one had attempted this before and that it required new algorithms and on-device processing to reliably detect app usage [27] [28]. Opal's method, using a local VPN, was reviewed by Apple and allowed on the App Store, whereas trying to directly access other apps' data without such indirection would likely violate guidelines. Opal's example shows that **sometimes the solution is to avoid Apple's Screen Time API altogether** when it proves too restrictive.

- **Qustodio's Approach:** Qustodio (a well-known cross-platform parental control app) uses a combination of techniques on iOS. According to their support docs, Qustodio's iOS app installs a **VPN profile** on the child's device which "sends your child's traffic through a local VPN" for monitoring [19]. They also use an **MDM profile** – evidenced by their installation instructions and user reports (e.g. a parent on Apple's forums asking how to prevent their teen from deleting the Qustodio MDM profile) [13]. The MDM profile allows Qustodio to enforce certain restrictions (like preventing app deletion or configuration changes) and possibly to gather the list of installed apps. With the bundle IDs from

that list, Qustodio can fetch app names and icons (likely via Apple's App Store lookup API) to display in the parent dashboard. This dual approach (VPN + MDM) is powerful but is only viable because Qustodio is marketed squarely for parental control – users willingly go through a complex setup for the sake of monitoring.

- **Jomo's Hybrid Method:** As mentioned, Jomo blends Apple's Screen Time data with its own estimation. Their help center explicitly states *"for privacy and security reasons, Apple doesn't allow access to Screen Time data in certain cases (like widgets), or the data isn't accurate enough for some features"* [20], which is why they have an "Estimated Screen Time" calculated by Jomo itself. This suggests Jomo might track screen-unlock time or app usage via an Accessibility service or by detecting when its own Screen Time interventions trigger. Jomo's advice to disable **Share Across Devices** implies they want to avoid Apple automatically merging data from multiple child devices (which can cause the child's phone to report usage from their iPad, etc., confusing the app) [22]. This highlights another nuance: even if you get Apple's data, it may include other devices under the same iCloud Screen Time if that feature is on, complicating per-device reporting. Jomo's solution is to ask the user to turn that off for clarity.

Conclusion

In the current state of iOS (as of 2025), **no perfect, straightforward method exists to collect a child device's app names and icons and send them to a parent's device using only public APIs**. Apple's Screen Time APIs deliberately abstract app identities for privacy, providing only tokens and system-rendered UI elements as a compromise [2] [6]. Developers have confronted this by either working within Apple's constraints – using on-device extensions and labels to display info without ever "grabbing" it – or by inventing out-of-band solutions (like local VPN monitoring, or MDM-based supervision) to bypass the need for Apple's data altogether.

Open-source and community knowledge confirms these challenges. Multiple Stack Overflow posts and Apple forum threads document that attempts to get bundle IDs or names from `FamilyActivitySelection` will fail [29] [6]. The officially blessed path is to use the data within the DeviceActivity Monitor/Report extensions or the shield UI, and nowhere else [6] [30]. On the other hand, **commercial apps** in the parental control space often play by a different rulebook: since they aim to provide a rich cross-device experience to paying customers, they leverage techniques that require extra setup (VPNs, profiles) but give them the needed data. These techniques have their own trade-offs (complex setup, potential performance impact, and the ever-present risk of Apple policy changes).

For a developer facing this problem, the **troubleshooting done by others** suggests a few options: - Use Apple's Screen Time API for what it's good at (enforcing limits, getting usage within an on-device UI context) and accept its privacy limitations for data sharing [2] [6]. - If needing actual app identifiers and icons, consider an MDM/supervised device approach, which allows querying installed apps (with user consent and Apple's approval) [15]. - Alternatively, implement a network-based usage logger (VPN) or other creative on-device tracking to independently identify apps by behavior [18] [10]. - Remember that any solution must comply with Apple's guidelines – using private APIs or extracting Screen Time data without permission can result in App Store rejection [16].

In conclusion, **the experiences of similar apps show that this is a non-trivial engineering hurdle**. Developers have often ended up with hybrid solutions, and many have lobbied Apple (via feedback and

forums) for better Screen Time API features. Until Apple relaxes the data restrictions or provides a parent->child data sharing mechanism, collecting app names/icons from a child's device will remain challenging. The best practice is to use the sanctioned frameworks as much as possible and supplement with carefully chosen workarounds (as seen in apps like Opal, Qustodio, and Jomo) to achieve the desired functionality. Each approach comes with maintenance overhead and potential fragility, but the community's shared knowledge can guide new developers in making informed trade-offs.

References

- Apple Developer Forums – Discussion of tokens and bundle identifiers [29] [3] [30]
- Stack Overflow – Q&A on retrieving app names from FamilyActivityPicker [6] [7]
- Frederik Riedel's blog on Screen Time API issues (One Sec app) [1] [2]
- Reddit iOSProgramming – Discussion of Screen Time API limitations and Opal's approach [10] [11]
- Opal Engineering Blog – "Building Opal's Screen Time Framework" (VPN-based solution) [17] [18]
- Qustodio Support FAQ – Use of VPN and MDM profile on iOS [19] [13]
- Jomo Help Center – Explanation of Estimated vs. Apple Screen Time data [20] [21]
- Apple Developer Documentation/WWDC – Screen Time/Family Controls framework overview [1] [2] (privacy design of tokens)
- Apple Developer Forums – Example code using DeviceActivityReport extension to get app names in extension [8]
- Stack Overflow – MDM for parental control and Apple's policy (historical context) [16] [31] .

---

[1] [2] Apple's Screen Time API has some major issues | riedel.wtf
https://riedel.wtf/state-of-the-screen-time-api-2024/

[3] [9] [29] [30] How to get the bundleIdentifier or... | Apple Developer Forums
https://developer.apple.com/forums/thread/782492

[4] [5] [8] [25] [26] Family Controls | Apple Developer Forums
https://developer.apple.com/forums/tags/family-controls?page=5&sortBy=lastUpdated

[6] [7] swiftui - How to retrieve the App name or bundle Identifier from the familyActivityPicker in swift - Stack Overflow
https://stackoverflow.com/questions/79610682/how-to-retrieve-the-app-name-or-bundle-identifier-from-the-familyactivitypicker

[10] [11] [12] About the Screen Time API. : r/swift
https://www.reddit.com/r/swift/comments/1jbuqg3/about_the_screen_time_api/

[13] Why do I get "Profile Installation Failed" when installing on iOS
https://help.qustodio.com/hc/en-us/articles/360005220618-Why-do-I-get-Profile-Installation-Failed-when-installing-on-iOS

[14] Qustodio is actually wild. And I need help with it. : r/helicopterparents
https://www.reddit.com/r/helicopterparents/comments/1il9spc/qustodio_is_actually_wild_and_i_need_help_with_it/

[15] [16] [31] ios - Apple MDM custom server for parental control app - Stack Overflow
https://stackoverflow.com/questions/35066433/apple-mdm-custom-server-for-parental-control-app

[17] [18] [27] [28] Building Opal's Screen Time Framework | Opal
https://www.opal.so/blog/opals-screen-time-framework

19 FAQs: Qustodio for iPhone and iPad
https://help.qustodio.com/hc/en-us/articles/360005220738-FAQs-Qustodio-for-iPhone-and-iPad

20 21 22 Estimated screen time is incorrect | Jomo Help Center
https://help.jomo.so/en/article/estimated-screen-time-is-incorrect-mhgq52/

23 GitHub - kingstinct/react-native-device-activity: Provides direct access to Apples Screen Time, Device Activity and Shielding APIs
https://github.com/kingstinct/react-native-device-activity

24 A Developer’s Guide to Apple’s Screen Time APIs (FamilyControls, ManagedSettings, DeviceActivity) | by Juliusbrussee | Medium
https://medium.com/@juliusbrussee/a-developers-guide-to-apple-s-screen-time-apis-familycontrols-managedsettings-deviceactivity-e660147367d7