ChatGPT

Maximizing Tracking Accuracy

- **Enable Screen Time entitlements & permissions.** Use Apple's *FamilyControls* APIs with the `com.apple.developer.family-controls` entitlement (must request Apple approval) [1] [2]. At runtime, call `AuthorizationCenter.shared.requestAuthorization(for: .individual)` and have the user select which apps/categories to monitor via `FamilyActivityPicker` [3] [4]. Without the entitlement or user consent, no usage data is available.
- **Use DeviceActivity frameworks for foreground usage.** Implement a **DeviceActivityMonitor** extension that schedules monitoring intervals (e.g. daily 0:00–23:59) and **DeviceActivityEvent** thresholds to accumulate time. In practice, set up frequent short thresholds (for example, every 5 minutes) so that `eventDidReachThreshold` fires repeatedly and you can tally seconds of foreground activity [5] [6]. (One known approach is dividing the day into two-hour slots with 5 min increments, yielding up to ~288 threshold events per day [5] [6].) Each time the threshold fires, add the interval (e.g. 5 min) to your usage total.
- **Fetch detailed data with DeviceActivityReport.** Also add a **DeviceActivityReport** extension (a SwiftUI view) to display per-app usage breakdown. In the extension's `makeConfiguration(representing:)`, the system provides `DeviceActivityResults` containing each app's total foreground time for the scheduled interval [7]. You can use this to show or summarize actual usage per app. (Note: the report runs in its own sandbox and can only display data; it cannot export it to your main app [8] [9].)

Platform Limitations (Non-MDM App)

- **Special entitlement required.** The Screen Time APIs are locked behind Apple review. You **must** obtain the Family Controls entitlement (`com.apple.developer.family-controls`) for each target that uses these APIs [2] [1]. Apps without this entitlement cannot use the Screen Time or FamilyControls frameworks.
- **User-selected scope only.** An app can only track usage of apps/categories **explicitly chosen by the user** via the FamilyActivityPicker [4]. You cannot programmatically enumerate all apps or inject code into other apps. The user's selection yields opaque tokens for apps/categories; the main app never sees actual bundle IDs or names (this is by design for privacy) [9]. To display app names or icons, you must do so within the DeviceActivityReport extension (e.g. using `Label(applicationToken)`).
- **No cross-app communication.** The DeviceActivityReport extension is sandboxed: it can't write to disk, send network requests, or communicate data to the main app [8]. All logic using the usage data must occur *within* the extension's SwiftUI view callbacks. In practice, this means your app can tell the user about usage, but it cannot import the raw usage numbers back into your app's code.
- **MDM and privacy constraints.** Unlike enterprise/MDM solutions, App-Store apps have **no direct background hooks** or hidden APIs for monitoring apps. The only approved method is via these Screen Time frameworks. (Note: older parental-control apps using MDM were banned unless they met strict guidelines.) Also, by default Screen Time only reports activity on the local device; with "Share Across Devices" enabled, it syncs multiple devices into one tally, which can skew single-device tracking – developers have reported double-counting if a user's devices are linked [10] [11].

Idle-Time Detection

- **No dedicated idle event.** iOS does not emit a “user is idle” signal through these APIs. Apple’s Screen Time logic simply stops counting when the device is not actively used (screen locked or off) [12]. By design, listening to audio, having the phone sit unlocked without touches, or locking the screen generates **no additional screen-time**. Inference is therefore limited: if your monitor sees no activity (no threshold events) for a long stretch, you may assume the user was idle, but the API gives only the opposite (active) signal.
- **Heuristic approaches only.** Some apps attempt to infer idle by observing gaps between usage reports, but with coarse resolution (tens of minutes at best). For example, if you fire 5-min thresholds, a gap where the final threshold of an interval doesn’t fire until much later suggests idle. But note that the **DeviceActivityMonitor**’s `intervalDidEnd` can fire even if the user is inactive (it simply marks the schedule end) [6], so interpreting that event as “idle” is unreliable. In short, **no official idle-detection API exists**; any idle inference must be application-level logic (e.g. comparing timestamps) outside the Screen Time framework.

Known Pitfalls & Inaccuracies

- **Reporting delays and latency.** Screen Time data is not real-time. Built-in iOS Screen Time can lag by minutes or hours as it processes usage [13]. Likewise, developers report that the DeviceActivityReport UI often loads slowly or even appears blank until refreshed [14]. Workarounds include caching yesterday’s data in advance or prompting the user to refresh.
- **Sandbox bugs and limits.** The DeviceActivityReport extension is severely sandboxed: it only has ~6 MB RAM [15] and no external I/O. Complex logic or decoding a lot of data inside the extension can crash it. Also, Apple notes (and devs confirm) that `intervalDidEnd` sometimes fails to fire on schedule [16] [17]. Always test over longer intervals (the *minimum* interval length is 15 minutes [18]) and be prepared for occasional missed callbacks.
- **Platform bugs.** Specific iOS versions have reported bugs. For instance, on iOS 17.6 many developers saw **double-counted time**: Safari usage and web categories both counted, inflating totals [11]. Also, the new “Share Across Devices” bug in iOS 17.6 caused monitors to include other devices’ usage even when turned off [10] [19]. Watch Apple’s release notes and developer forums for such issues, and advise users to disable cross-device sync if totals seem wrong.
- **Resource limits.** You can schedule at most ~20 active `DeviceActivityName` schedules at once [20]. If you need finer granularity (e.g. per-hour schedules), you must carefully manage and renew schedules. Registering many schedules/events is also slow and should be done on a background thread [20]. Exceeding these limits will silently fail or crash.
- **Data privacy features.** By default, Screen Time omits logged-out or non-owned apps. If parental restrictions or content filters (“Downtime”, etc.) are active, Screen Time may not even count certain categories at all. There is no way around this in App-Store apps – it’s built into the OS.
- **Extension-only output.** Remember that **all usage data can only be presented via the DeviceActivityReport view**. You cannot use `print()`, notifications, or file output to capture the data [8]. Debug by inspecting logs in the extension (via Console) or by building the output into the SwiftUI view itself (as in sample projects [7]).

Update Intervals & Thresholds

- **Scheduled intervals.** DeviceActivity schedules run on time-of-day ranges you define. Callbacks occur at *interval start* (`intervalDidStart`), at each *threshold event* (`eventDidReachThreshold`), and at *interval end* (`intervalDidEnd`) [21]. Use `DeviceActivitySchedule` to repeat daily/weekly. The **minimum interval duration is 15 minutes** [18], so you cannot schedule, say, a 1-minute total block.
- **Threshold granularity.** Each `DeviceActivityEvent` uses a `DateComponents` threshold (hour/minute). In practice, you can use very fine thresholds (e.g. every minute or 5 minutes) to improve resolution. Developers have commonly used 5-minute increments to estimate usage [5]. (Apple does not explicitly forbid 1-minute thresholds, but each extra threshold consumes part of the 20-schedule limit.) The trade-off is that shorter thresholds mean more frequent wake-ups and more counters to manage.
- **No immediate app-switch event.** The Screen Time API does *not* fire an event on every app switch. You only get notifications at the boundaries you schedule. Thus, very short switching (e.g. a 2-minute app use) might not trigger a threshold event if your next threshold is at 5 minutes. To mitigate this, set your first threshold shortly after the interval starts (`includesPastActivity: true` can cause an immediate event if usage already exceeds it) or use frequent warnings.
- **Updates in DeviceActivityReport.** The report view uses a `DeviceActivityFilter` (by date/device) to query data. For example, a daily filter will cover midnight-to-midnight. The report data is fetched when the extension's view appears or when the system refreshes it; there is no continuous "push" update. In practice, present the report view when needed (e.g. on demand or scheduled reminder).

Sources: Apple's Screen Time API documentation and examples, WWDC sessions, and developer reports [2] [9] [5] [6] [15] [12] [14] [11].

---

[1] Using Screen Time API to block apps for a specified time
http://pedroesli.com/2023-11-13-screen-time-api/

[2] [3] 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

[4] [5] [6] [17] [20] A curious way to grab Screen Time data on iOS · matteing.com · matteing.com
https://matteing.com/posts/a-curious-way-to-grab-screen-time-data-on-ios

[7] [8] Creating an iOS Screen Time Tracking App Using SwiftUI and Apple's DeviceActivity Framework | by Muhammad Danish Qureshi | Medium
https://medium.com/@danisharfin1/creating-an-ios-screen-time-tracking-app-using-swiftui-and-apples-deviceactivity-framework-e999c6f37930

[9] ios - Swift using Family Controls to limit apps and get name of app - Stack Overflow
https://stackoverflow.com/questions/79286261/swift-using-family-controls-to-limit-apps-and-get-name-of-app

[10] [11] [19] DeviceActivityMonitor is overcount... | Apple Developer Forums
https://developer.apple.com/forums/thread/763542

[12] [13] Understanding Screen Time on iPhone: What Really Counts?
https://www.airdroid.com/ios-parental/what-counts-as-screen-time-iphone/

[14] Help with Apple's new Screen Time API being super laggy? : r/iOSProgramming
https://www.reddit.com/r/iOSProgramming/comments/12m10eg/help_with_apples_new_screen_time_api_being_super/

[15] [16] [18] [21] Time After (Screen) Time, part 3. The Device Activity Monitor Extension. | by letvar | Medium
https://letvar.medium.com/time-after-screen-time-part-3-the-device-activity-monitor-extension-284da931391b