UIApplication Background Task Notes

This thread has been locked by a moderator.

26k

The `UIApplication` background task mechanism allows you to prevent your app from being suspended for short periods of time. While the API involved is quite small, there's still a bunch of things to watch out for.

The name *background task* is somewhat misappropriate. Specifically, `beginBackgroundTask(expirationHandler:)` doesn't actually start any sort of background task, but rather it tells the system that *you* have started some ongoing work that you want to continue even if your app is in the background. You still have to write the code to create and manage that work. So it's best to think of the background task API as raising a "don't suspend me" assertion.

You must end every background task that you begin. Failure to do so will result in your app being killed by the watchdog. For this reason I recommend that you attach a name to each background task you start (by calling `beginBackgroundTask(withName:expirationHandler:)` rather than `beginBackgroundTask(expirationHandler:)`). A good name is critical for tracking down problems when things go wrong.

IMPORTANT Failing to end a background task is the number one cause of background task problems on iOS. This usually involves some easy-to-overlook error in bookkeeping that results in the app begining a background task and not ending it. For example, you might have a property that stores your current background task identifier (of type `UIBackgroundTaskIdentifier`). If you accidentally creates a second background task and store it in that property without calling `endBackgroundTask` on the identifier that's currently stored there, the app will 'leak' a background task, something that will get it killed by the watchdog.

Background tasks can end in one of two ways:

- When your app has finished doing whatever it set out to do.
- When the system calls the task's expiry handler.

Your code is responsible for calling `endBackgroundTask(_:)` in both cases.

All background tasks must have an expiry handler that the system can use to 'call in' the task. The background task API allows the system to do that at any time.

Your expiry handler is your opportunity to clean things up. It should not return until everything is actually cleaned up. It must run quickly, that is, in less than a second or so. If it takes too long, your app will be killed by the watchdog.

Your expiry handler is called on the main thread.

It is legal to begin and end background tasks on any thread, but doing this from a secondary thread can be tricky because you have to coordinate that work with the expiry handler, which is always called on the main thread.

The system puts strict limits on the total amount of time that you can prevent suspension using background tasks. On current systems you can expect about 30 seconds.

IMPORTANT I'm quoting these numbers just to give you a rough idea of what to expect. The target values have changed in the past and may well change in the future, and the amount of time you actually get depends on the state of the system. The thing to remember here is that the exact value doesn't matter as long as your background tasks have a functional expiry handler.

You can get a rough estimate of the amount of time available to you by looking at `UIApplication`'s `backgroundTimeRemaining` property.

IMPORTANT The value returned by `backgroundTimeRemaining` is an estimate and can change at any time. You must design your app to function correctly regardless of the value returned. It's reasonable to use this property for debugging but we strongly recommend that you avoid using as part of your app's logic.

IMPORTANT Basing app behaviour on the value returned by `backgroundTimeRemaining` is the number two cause of background task problems on iOS.

The system does not guarantee *any* background task execution time. It's possible (albeit unlikely, as covered in the next point) that you'll be unable to create a background task. And even if you do manage to create one, its expiry handler can be called at any time.

`beginBackgroundTask(expirationHandler:)` can fail, returning `UIBackgroundTaskInvalid`, to indicate that you the system is unable to create a background task. While this was a real possibility when background tasks were first introduced, where some devices did not support multitasking, you're unlikely to see this on modern systems.

The background time 'clock' only starts to tick when the background task becomes effective. For example, if you start a background task while the app is in the foreground and then stay in the foreground, the background task remains dormant until your app moves to the background. This can help simplify your background task tracking logic.

The amount of background execution time you get is a property of your app, not a property of the background tasks themselves. For example, starting two background task in a row won't give you 60 seconds of background execution time.

Notwithstanding the previous point, it can still make sense to create multiple background tasks, just to help with your tracking logic. For example, it's common to create a background task for each job being done by your app, ending the task when the job is done.

Do not create too many background tasks. How many is too many? It's absolutely fine to create tens of background tasks but creating thousands is not a good idea.

IMPORTANT iOS 11 introduced a hard limit on the number of background task assertions a process can have (currently about 1000, but the specific value may change in the future). If you see a crash report with the exception code 0xbada5e47, you've hit that limit.

Note The practical limit that you're most likely to see here is the time taken to call your expiry handlers. The watchdog has a strict limit (a few seconds) on the total amount of time taken to run background task expiry handlers. If you have thousands of handlers, you may well run into this limit.

If you're working in a context where you don't have access to `UIApplication` (an app extension or on watchOS) you can achieve a similar effect using the `performExpiringActivity(withReason:using:)` method on `ProcessInfo`.

If your app 'leaks' a background task, it may end up being killed by the watchdog. This results in a crash report with the exception code 0x8badf00d ("ate bad food").

IMPORTANT A leaked background task is *not* the only reason for an 0x8badf00d crash. You should look at the backtrace of the main thread to see if the main thread is stuck in your code, for example, in a synchronous networking request. If, however, the main thread is happily blocked in the run loop, a leaked background task should be your primary suspect.

Prior to iOS 11 information about any outstanding background tasks would appear in the resulting crash report (look for the text `BKProcessAssertion`). This information is not included by iOS 11 and later, but you can find equivalent information in the system log.

The system log is very noisy so it's important that you give each of your background tasks an easy-to-find name.

For more system log hints and tips, see Your Friend the System Log.

---

iOS 13 introduced the Background Tasks framework. This supports two type of requests:

- The `BGAppRefreshTaskRequest` class subsumes UIKit's older background app refresh functionality.
- The `BGProcessingTaskRequest` class lets you request extended background execution time, typically overnight.

WWDC 2020 Session 10063 Background execution demystified is an excellent summary of iOS's background execution model. Watch it, learn it, love it!

For more background execution hints and tips, see Background Tasks Resources.

Share and Enjoy
—
Quinn "The Eskimo!" @ Developer Technical Support @ Apple
`let myEmail = "eskimo" + "1" + "@" + "apple.com"`

Revision History

- **2022-06-08** Corrected a serious error in the discussion of `BGProcessingTaskRequest`. Replaced the basic system log info with a reference to Your Friend the System Log. Added a link to Background Tasks Resources. Made other minor editorial changes.
- **2021-02-27** Fixed the formatting. Added a reference to the Background Tasks framework and the *Background execution demystified* WWDC presentation. Minor editorial changes.
- **2019-01-20** Added a note about changes in the iOS 13 beta. Added a short discussion about beginning and ending background tasks on a secondary thread.
- **2018-02-28** Updated the task name discussion to account for iOS 11 changes. Added a section on how to debug 'leaked' background tasks.
- **2017-10-31** Added a note about iOS 11's background task limit.
- **2017-09-12** Numerous updates to clarify various points.
- **2017-08-17** First posted.

Background Tasks

Reply

Posted 5 years ago by eskimo

Add a Comment