Diving into Doze Mode for Developers

Paul Turner
Jul 14, 2016 • Android

Recent versions of Android have focused on improving device performance, and one of the biggest improvements came with the addition of Doze Mode in Android Marshmallow and Nougat. In a previous post, we talked about background schedulers in Android, briefly mentioning the effects of Doze Mode in Android Marshmallow. This post will cover Doze Mode in more detail, the changes that are coming with Android Nougat and what you can do to correctly adapt to Doze Mode.

What is Doze Mode?

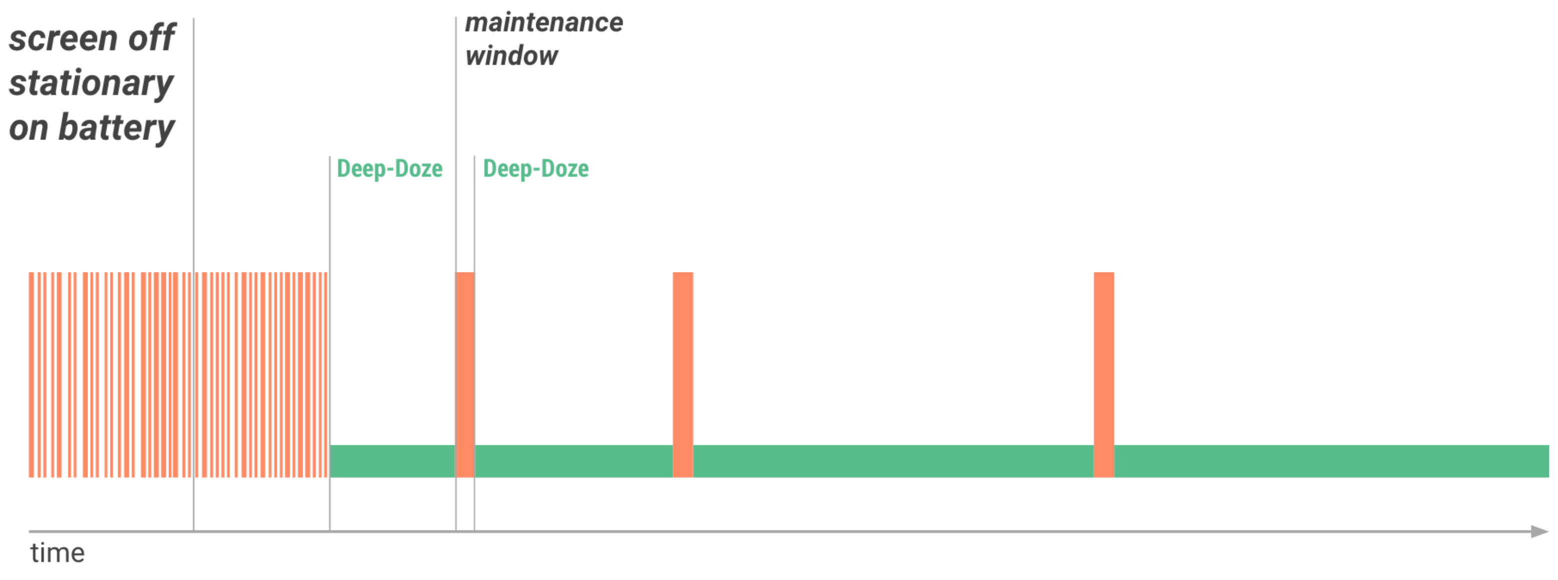
Doze Mode focuses on improving battery life on Android by cutting down on background network activity when the device is not in use. Background updates by apps are instead grouped into controlled maintenance windows, allowing for data freshness when the user comes back to their phone and exits Doze Mode.

Developers usually have control over opting into changes to the Android operating system by incrementing the target SDK level of their applications. However, in order for Doze to have a real impact on the battery life of the device, Doze Mode changes are applied to all apps regardless of target SDK level. The good news is that if you are using the framework properly there probably won't be much you will need to change in your application. This blog post will cover when Doze is activated, what restrictions are applied, how to work around Doze (if necessary), and how to test your app for Doze effects.

When Does Android Doze?

The first concepts we need to understand are the various "stages" of Doze Mode and when they will be active on devices in Android Nougat. The main stage of Doze (referred to as Deep-Doze in this article) was introduced with Android Marshmallow and is activated when the device is not charging, the screen is off, and the device is completely stationary. Android checks for when these conditions were met, waits roughly 30 minutes to be sure the phone does not change conditions, and then jumps into the Deep-Doze battery saving mode. When any of these conditions change, the device will exit Deep-Doze Mode and restrictions will be lifted. In order to still give users a pleasant experience when they come back to their phone, Deep-Doze provides short maintenance windows where apps can sync up their latest data. These maintenance windows are spaced out further and further the longer a user is in

Deep-Doze, increasing power savings. The following diagram visualizes these effects in Android Marshmallow:

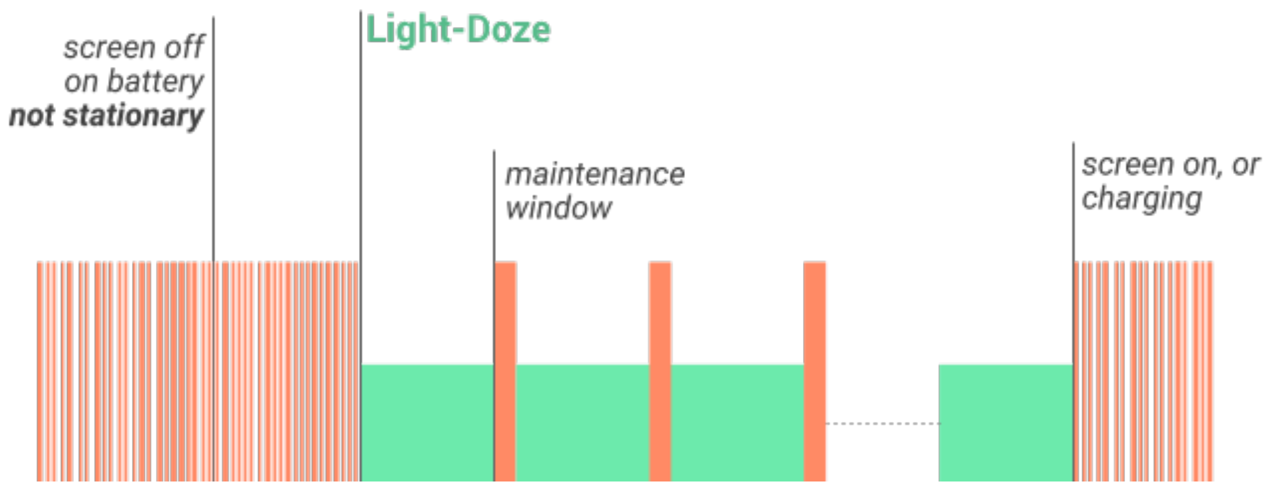


The orange bars represent network activity on the device, and the green section when Deep-Doze restrictions are being applied. The “barcode” appearance of the orange bars represents network activity launching at seemingly random points in time from various applications on the phone. However, in Deep-Doze Mode there are more controlled maintenance windows in which apps background work is bundled together to run in a more compact period of time.

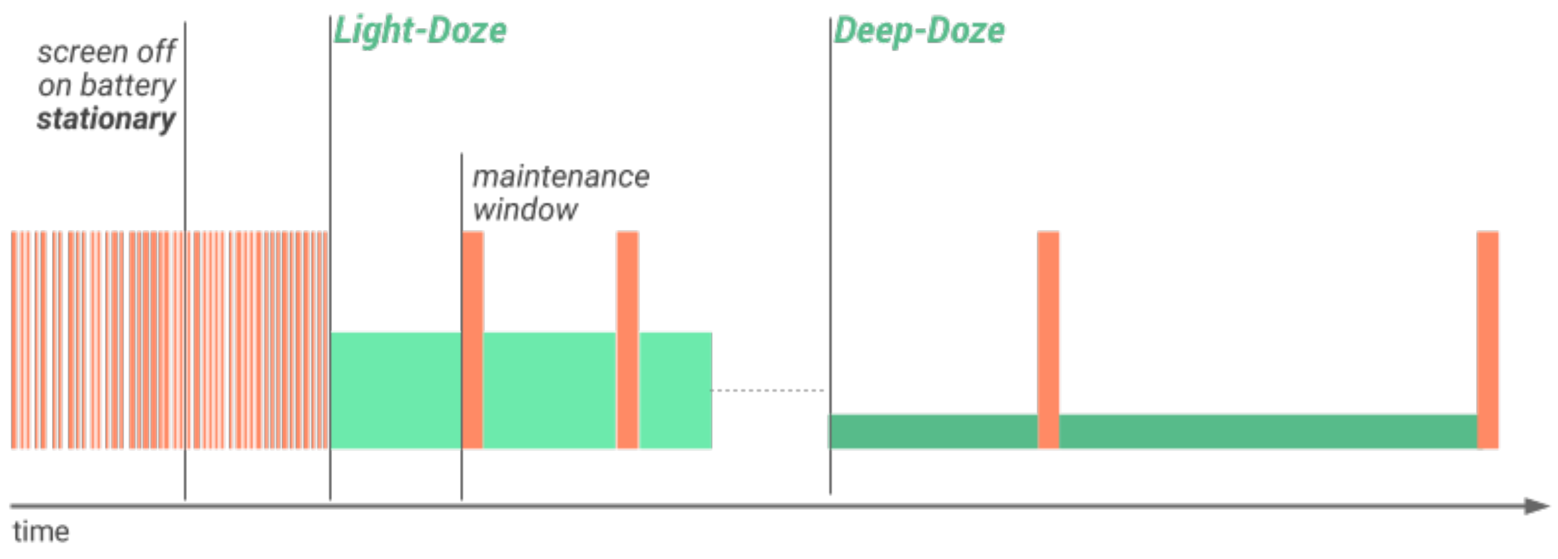
Android Nougat changes this formula with the addition of a more lightweight version of Doze that does not require the device to be completely stationary or wait as long to activate. I will be referring to this lightweight version of Doze as Light-Doze here. Light-Doze starts to kick in shortly after both the screen is off and the device is not charging, waiting only a couple minutes before applying the restrictions just to make sure the user has stopped using their phone. Light-Doze has fewer restrictions placed for battery savings, so the period between data-syncing maintenance windows is shorter. The goal is to provide a quicker step to Doze Mode with a smaller impact before initiating Deep-Doze Mode.

While in Light-Doze Mode, if the device has remained stationary, the system will drop the device into Deep-Doze Mode and its corresponding maintenance windows. Similarly, Light-Doze will be interrupted when the device is plugged in or the screen is turned on. It is also important to note that Deep-Doze can be interrupted and stepped up to Light-Doze if the device stops being stationary but still not charging with the screen off.

Google has provided a couple of diagrams to help visualize some of these situations, the first of which visualizes Light-Doze becoming active and inactive:



In this diagram we are stepping into Light-Doze after a couple of minutes, and the maintenance windows are more frequent than what we saw with Deep-Doze. This diagram shows Light-Doze as a slightly taller and lighter shade of green to signify the lesser restrictions compared to Deep-Doze. The more interesting situation is how the device will step into Deep-Doze with Nougat:



Even though the device is meeting all three of the criteria (screen off, on battery and stationary) the OS will first step into Light-Doze while it confirms that the device has indeed been stationary for enough time.

This time period matches the second group of “barcode” activity that we saw in our first Deep-Doze image from Android Marshmallow. Once that period has passed, Deep-Doze Mode is initiated and we start to see the greater battery-saving restrictions applied.

What does Doze do?

So now that we have covered when Deep-Doze and Light-Doze come into effect, we need to take a look at the actual limitations it places on apps in the device. Light-Doze behaviors are a subset of Deep-Doze behaviors— network access is cut off for all applications and Jobs will be deferred from JobScheduler and Syncs from Sync-Adapters.

These restrictions are lifted when maintenance mode begins, and any deferred Jobs and Syncs will be started at that point. If you are using AlarmManager for your recurring background work those will continue to kick off, but it is important to note that you will not have network access in that time. If this is a deal breaker for you, it might be time to migrate your background work into JobScheduler so that you only start up when the network is available in the maintenance windows.

When the device enters Deep-Doze Mode, a couple of additional restrictions are applied. AlarmManager alarms will be deferred just like Jobs and Syncs are in Light-Doze—held onto until thedevice goes into a maintenance window or exits Doze entirely. The operating system will also ignore any wakelocks attempted by the application, which means work must be completed within the maintenance window as your wakelock will ignored when it jumps back into Doze. Finally, Deep-Doze will also stop the system from making Wifi and GPS scans, reducing location events from the device detecting WiFi hotspots.

Interrupting Doze

So now that you understand the effects Doze can have on your app, you may be thinking, "How will the user survive if I'm not allowed to run background work and notify the user of events whenever I want?!"

This shouldn't be the case, as Light-Doze has minimal impact given that the maintenance windows occur frequently and Deep-Doze only affects your app when the user is not using their device at all (think sitting on a desk overnight, or left behind in another room while cooking). That being said, there are still some situations where you need to get around Doze's effects, and there are a couple of options for what you can do.

If you need to make sure a notification is delivered immediately, then Firebase Cloud Messaging (FCM, formerly known as Google Cloud Messaging) can do so without you having to change your app's source code.

FCM has the ability to send a notification to the device while in Doze Mode simply by setting the message to a high-priority message. Google warns that this should be reserved for "only if the message is time-critical and requires the user's immediate interaction," as this will interrupt Doze Mode and grant temporary access to network services and partial wakelocks for your app only.

If you want to have some background work run while the device is in Doze Mode and not yet in a maintenance window, then AlarmManager also has some options for your app. The methods `AlarmManager.setAndAllowWhileIdle()` and `AlarmManager.setExactAndAllowWhileIdle()` will allow the PendingIntent to fire even if the device is in Doze Mode ("idle" is considered to be the time in which Doze Mode is active).

It's important to note that this will not lift network restrictions and will give you a 10-second window to capture further wake locks in which to complete your work. Again, Android documentation warns this should only be used in when absolutely critical, as it will affect the users battery by interrupting Doze Mode and this will be reflected in the battery stats.

`AlarmManager.setAlarmClock()` will also ignore Doze Mode and fire as normal, with the added effect of exiting Doze Mode completely shortly before your alarm fires (allowing the app to have fresh data before the user picks up their phone). This should not be used for your normal background work alarms, as `AlarmManager.setAlarmClock()` is intended to be used for what the user perceives as an alarm clock—a scheduled time at which the phone will alert the user (not just a calendar event). An icon is even displayed in the status bar as a physical alarm clock face, and many lockscreens will also inform the user of when the next alarm is scheduled.

Finally, if high-priority FCM messages and while-idle alarms are not enough, you have the option of having the user opt you into a whitelist of apps that are partially exempt from Doze restrictions. A whitelisted app can access the network and hold partial wakelocks, but the rest of Doze restrictions will still apply (notably the deferring of Jobs, Syncs and Alarms). There is no way to programmatically toggle your app to be part of this whitelist, but you can prompt the user to add you in one of two ways.

You can launch an Intent with the ACTION_IGNORE_BATTERY_OPTIMIZATION_SETTINGS action and take the user to the Battery Optimization page where they can search through a list of all apps and turn off optimizations for your app.

Or, if you have requested the REQUEST_IGNORE_BATTERY_OPTIMIZATIONS permission, you can directly prompt the user with a dialog to turn off optimizations for your app without needing the user to search for your app by launching an Intent with the ACTION_REQUEST_IGNORE_BATTERY_OPTIMIZATIONS action.

Note that in the Android Nougat Developer Preview 4, the Settings app requires a force close to reflect changes made from this dialog. Also note that these are only for extreme cases and are discouraged by Google. The Doze documentation states: “Note: Google Play policies prohibit apps from requesting direct exemption from Power Management features in Android 6.0+ (Doze and App Standby) unless the core function of the app is adversely affected.”

That said, be sure your application requires whitelisting before prompting the user. Android documentation contains some guidance found here.

One last case which is not mentioned in the Android documentation is an app using a foreground service. Any process using a foreground service is exempt from Doze Mode effects, which is key for long-running applications that you may want to run even if the user has put their phone down for a long period of time—like a music app, for example.

There is, however, a bug in Android Marshmallow which requires a foreground service to be in a separate process from all other activity code, documented here. It is planned to be fixed in Nougat.

The important takeaway here is that in most cases you shouldn’t need to interrupt or get around Doze, thanks to the provided maintenance windows and the fact that the user is not actively using their phone.

The next step is to verify that your app works correctly under Doze without needing to use any of the workarounds mentioned above.

Testing Doze on your device

Now that you have more context on Doze and what effect it has on your device, you may want to test your app to make sure your device is behaving properly. Fortunately, Android has made it easy to do so without having to wait around for your device to settle into Doze Mode by using a couple of adb commands. Listed below are the steps for testing your device. Note that these assume a device running the Android Nougat Developer Preview 4 or later for access to Light-Doze:

1. Connect the device to your development machine and install your app.
2. Run your app and leave it active.
3. Shut off the device screen. (The app remains active.)
4. Force the system to cycle through Doze Modes by running the following commands:

```
$ adb shell dumpsys battery unplug
$ adb shell dumpsys deviceidle step [light|deep]
```

The first command will allow the system to think it is unplugged for you to continue testing while connected to USB for further adb commands (otherwise Doze Mode will never activate). The second command will step your device through each of the checks before being idle, printing out the current state after each step. By default, the command will step through the full Doze states and not bother with Light-Doze. You can add `light` or `deep` at the end to specify that you'd like to step through the corresponding Doze Modes. Once the device reaches `IDLE`, the step function will go back and forth between `IDLE` and `IDLE_MAINTENANCE` with each step.

You can query the current state at any point by calling:

```
$ adb shell dumpsys deviceidle get [light|deep|force|screen|charging|network]
```

As of the Android Nougat Developer Preview 4, Doze has the following states:

```
Light: ACTIVE -> IDLE -> IDLE_MAINTENANCE -> OVERRIDE
Deep: ACTIVE -> IDLE_PENDING -> SENSING -> LOCATING -> IDLE -> IDLE_MAINTENANCE
```

The `OVERRIDE` state is activated when Deep-Doze is either in the `IDLE` or `IDLE_MAINTENANCE` modes because Light-Doze has no effect once the device has gone into Deep-Doze Mode. The device will exit Doze when you turn the screen back on, but be sure to run the following command so that your device correctly displays your battery percentage and USB connection state after you are done testing:

```
$ adb shell dumpsys battery reset
```

More Information

I've written a quick little logging app that will display some of the effects of Doze Mode on Jobs and Alarms. The app will simply log out the current state when a Job/Alarm is kicked off, as well as logging when the device has stepped into idle mode. All of the log statements are written to disk so that you can view them in the app if you wish to test your device in real world usage. Otherwise, simply keep track of the log statements in the device log.

Looking for more information related to Doze? Here are some resources I found helpful in my dive into Doze:

- Doze Documentation: Explains Doze Mode and the related restrictions App Standby applies to unused applications on the phone.

- Doze Behaviour Changes: Changes to Doze Mode coming in Android Nougat.
- Android Nougat: More dozing without more code A brief video from the Google Developer Relations team covering the changes coming in Android Nougat.
- DeviceIdleController.java: Internal hidden class the system uses to drive through the different states of Doze Mode.
- Doze Google+ Post: Quick summary of Doze Mode by Joanna Smith of the Google Developer Relations team.
- FCM/GCM and Doze: Google Blog post covering how Google Cloud Messaging handles Doze in Android Marshmallow.

Hopefully I've demystified what Doze brings to the table and the new optimizations added in Android Nougat. I am a huge fan of what Doze Mode has done for my device's battery life, and I can't wait to see what comes next!

MORE BY
Paul Turner

Considering building an app?

See how our approach stands out from the crowd and helps your company meet its goals.

DISCOVER

RELATED POSTS:

Android Security for Developers
Bolot Kerimbaev

Perfecting Custom Typography in Android
Matt Raufman

Recent Comments

14 Comments **bignerdranch.com** 1 **Login**

Recommend 7 Share Sort by Oldest

Join the discussion…

Andranik Azizbekyan • 9 months ago
Thanks, very informative post! Bookmarked.
2 • Reply • Share ›

Avatar This comment was deleted.

Paul Turner Guest • 8 months ago
The transition from light doze to normal doze will begin once the device is motionless for some time, Google has said in the order of tens of minutes. After that it will do a couple more checks to make sure nothing is happening and jump into normal doze. You could get a sense for exact timing by exploring the DeviceIdleController (https://android.googlesourc... which has the states that it will jump through before getting into normal doze.
• Reply • Share ›

Avatar This comment was deleted.

Paul Turner Guest • 8 months ago
From what I've gathered, background activity won't have an effect on delaying going into full doze mode. You can have background activity interrupt it briefly though using the methods given in the "Interrupting Doze Mode" section. But I do not think they will delay doze mode unless they turn on the screen.
• Reply • Share ›

Krishnan Marimuthu • 4 months ago
Hi,
When we receive a GCM Notification how long the app has network access ? Can we prevent the app from entering doze quickly by holding a wakelock ?
1 • Reply • Share ›

Krishnan Marimuthu Krishnan Marimuthu • 4 months ago
I found the app has network access for 10 seconds. When I checked deviceidle using dumpsys, I found max_temp_app_whitelist_duration as 5 minutes. Can we somehow increase the 10 second timeout ?
• Reply • Share ›

Paul Turner Krishnan Marimuthu • 4 months ago
Hi there Krishnan,
Good digging for more information, unfortunately I don't have much experience with increasing the 10 second timeout. Wakelocks are not allowed unless your device is whitelisted.
Sorry I'm not more help here!
• Reply • Share ›

Krishnan Marimuthu Paul Turner • 4 months ago
Do you know the any official google forum to clarify these things ? When I googled lots of android forums popped up and couldn't find the official one
• Reply • Share ›

Paul Turner Krishnan Marimuthu • 4 months ago
Best I can think of is filing a bug if you think there is information that doesn't match. Otherwise there is no official google/android forum that you can get answers straight from developers that I know of. Your best bet would be a well explained question on stackoverflow, or some of the G+ communities that have some of the google developer advocates.
• Reply • Share ›

shubham • 4 months ago
C:\Users>adb shell dumpsys deviceidle get deep
Unknown command: get
• Reply • Share ›

Paul Turner shubham • 4 months ago
This will happen if you are running on a device that is not Nougat and above I believe. If you are testing a marshmallow