Android Developers

Optimizing for Doze and App Standby

Starting from Android 6.0 (API level 23), Android introduces two power-saving features that extend battery life for users by managing how apps behave when a device is not connected to a power source. *Doze* reduces battery consumption by deferring background CPU and network activity for apps when the device is unused for long periods of time. *App Standby* defers background network activity for apps with which the user has not recently interacted.

Doze and App Standby manage the behavior of all apps running on Android 6.0 or higher, regardless whether they are specifically targeting API level 23. To ensure the best experience for users, test your app in Doze and App Standby modes and make any necessary adjustments to your code. The sections below provide details.

In this document

Understanding Doze

If a user leaves a device unplugged and stationary for a period of time, with the screen off, the device enters Doze mode. In Doze mode, the system attempts to conserve battery by restricting apps' access to network and CPU-intensive services. It also prevents apps from accessing the network and defers their jobs, syncs, and standard alarms.

Periodically, the system exits Doze for a brief time to let apps complete their deferred activities. During this *maintenance window*, the system runs all pending syncs, jobs, and alarms, and lets apps access the network.

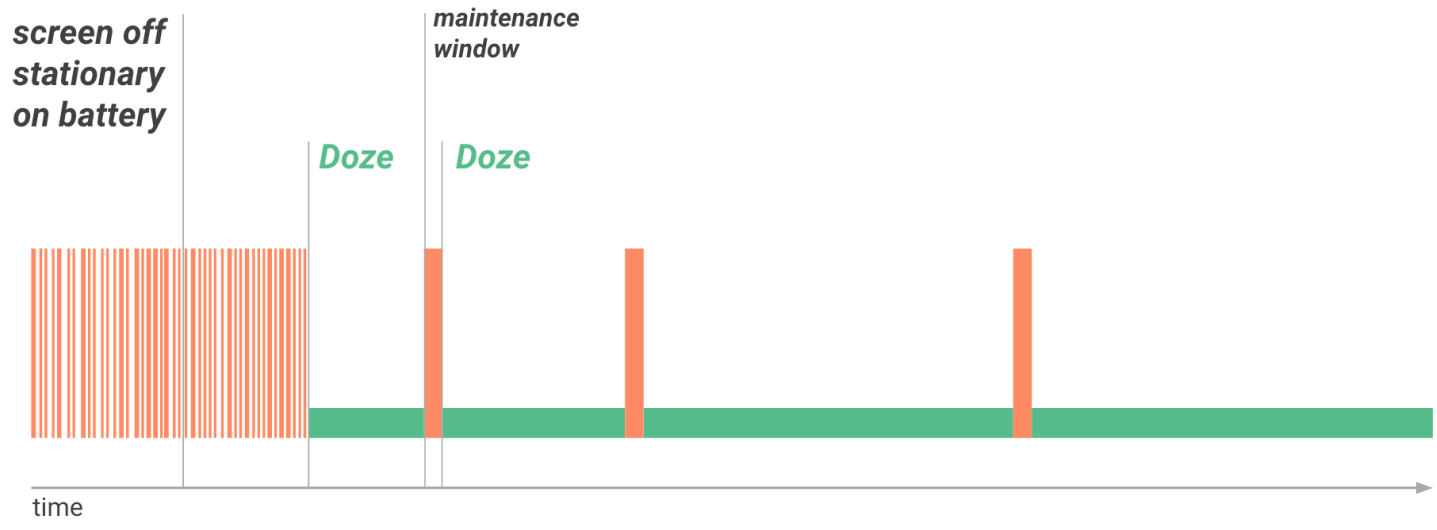

Figure 1. Doze provides a recurring maintenance window for apps to use the network and handle pending activities.

At the conclusion of each maintenance window, the system again enters Doze, suspending network access and deferring jobs, syncs, and alarms. Over time, the system schedules maintenance windows less and less frequently, helping to reduce battery consumption in cases of longer-term inactivity when the device is not connected to a charger.

As soon as the user wakes the device by moving it, turning on the screen, or connecting a charger, the system exits Doze and all apps return to normal activity.

Doze restrictions

The following restrictions apply to your apps while in Doze:

- Network access is suspended.
- The system ignores wake locks (https://developer.android.com/reference/android/os/PowerManager.WakeLock.html).
- Standard `AlarmManager` (https://developer.android.com/reference/android/app/AlarmManager.html) alarms (including `setExact()` (https://developer.android.com/reference/android/app/AlarmManager.html#setExact(int,

`long, android.app.PendingIntent))` and `setWindow()` (https://developer.android.com/reference/android/app/AlarmManager.html#setWindow(int, long, long, android.app.PendingIntent))) are deferred to the next maintenance window.

  - If you need to set alarms that fire while in Doze, use `setAndAllowWhileIdle()` (https://developer.android.com/reference/android/app/AlarmManager.html#setAndAllowWhileIdle(int, long, android.app.PendingIntent)) or `setExactAndAllowWhileIdle()` (https://developer.android.com/reference/android/app/AlarmManager.html#setExactAndAllowWhileIdle(int, long, android.app.PendingIntent)).
  - Alarms set with `setAlarmClock()` (https://developer.android.com/reference/android/app/AlarmManager.html#setAlarmClock(android.app.AlarmManager.AlarmClockInfo, android.app.PendingIntent)) continue to fire normally — the system exits Doze shortly before those alarms fire.

- The system does not perform Wi-Fi scans.
- The system does not allow sync adapters (https://developer.android.com/reference/android/content/AbstractThreadedSyncAdapter.html) to run.
- The system does not allow `JobScheduler` (https://developer.android.com/reference/android/app/job/JobScheduler.html) to run.

Adapting your app to Doze

Doze can affect apps differently, depending on the capabilities they offer and the services they use. Many apps function normally across Doze cycles without modification. In some cases, you must optimize the way that your app manages network, alarms, jobs, and syncs. Apps should be able to efficiently manage activities during each maintenance window.

Doze is particularly likely to affect activities that `AlarmManager` (https://developer.android.com/reference/android/app/AlarmManager.html) alarms and timers manage, because alarms in Android 5.1 (API level 22) or lower do not fire when the system is in Doze.

To help with scheduling alarms, Android 6.0 (API level 23) introduces two new `AlarmManager` (https://developer.android.com/reference/android/app/AlarmManager.html) methods: `setAndAllowWhileIdle()` (https://developer.android.com/reference/android/app/AlarmManager.html#setAndAllowWhileIdle(int, long, android.app.PendingIntent)) and `setExactAndAllowWhileIdle()` (https://developer.android.com/reference/android/app/AlarmManager.html#setExactAndAllowWhileIdle(int, long, android.app.PendingIntent)). With these methods, you can set alarms that will fire even if the device is in Doze.

> **Note:** Neither `setAndAllowWhileIdle()` (https://developer.android.com/reference/android/app/AlarmManager.html#setAndAllowWhileIdle(int, long, android.app.PendingIntent)) nor `setExactAndAllowWhileIdle()` (https://developer.android.com/reference/android/app/AlarmManager.html#setExactAndAllowWhileIdle(int, long, android.app.PendingIntent)) can fire alarms more than once per 9 minutes, per app.

The Doze restriction on network access is also likely to affect your app, especially if the app relies on real-time messages such as tickles or notifications. If your app requires a persistent connection to the network to receive messages, you should use Firebase Cloud Messaging (FCM) (#using_fcm) if possible.

To confirm that your app behaves as expected with Doze, you can use adb commands to force the system to enter and exit Doze and observe your app's behavior. For details, see Testing with Doze and App Standby (#testing_doze_and_app_standby).

Doze checklist

- If possible, use FCM for downstream messaging .
- If your users must see a notification right away, make sure to use an FCM high priority message .
- Provide sufficient information within the initial message payload , so subsequent network access is unnecessary.
- Set critical alarms with `setAndAllowWhileIdle()` and `setExactAndAllowWhileIdle()`.
- Test your app in Doze.

Understanding App Standby

App Standby allows the system to determine that an app is idle when the user is not actively using it. The system makes this determination when the user does not touch the app for a certain period of time and none of the

following conditions applies:

- The user explicitly launches the app.
- The app has a process currently in the foreground (either as an activity or foreground service, or in use by another activity or foreground service).

  > **Note:** You should only use a foreground service (https://developer.android.com/guide/components/services.html#Foreground) for tasks the user expects the system to execute immediately or without interruption. Such cases include uploading a photo to social media, or playing music even while the music-player app is not in the foreground. You should not start a foreground service simply to prevent the system from determining that your app is idle.

- The app generates a notification that users see on the lock screen or in the notification tray.
- The app is an active device admin app (for example, a device policy controller (https://developers.google.com/android/work/build-dpc)). Although they generally run in the background, device admin apps never enter App Standby because they must remain available to receive policy from a server at any time.

When the user plugs the device into a power supply, the system releases apps from the standby state, allowing them to freely access the network and to execute any pending jobs and syncs. If the device is idle for long periods of time, the system allows idle apps network access around once a day.

Using FCM to Interact with Your App While the Device is Idle

Firebase Cloud Messaging (FCM) (https://firebase.google.com/docs/cloud-messaging) is a cloud-to-device service that lets you support real-time downstream messaging between backend services and apps on Android devices. FCM provides a single, persistent connection to the cloud; all apps needing real-time messaging can share this connection. This shared connection significantly optimizes battery consumption by making it unnecessary for multiple apps to maintain their own, separate persistent connections, which can deplete the battery rapidly. For this reason, if your app requires messaging integration with a backend service, we strongly recommend that you **use FCM if possible**, rather than maintaining your own persistent network connection.

FCM is optimized to work with Doze and App Standby idle modes by means of high-priority FCM messages (https://firebase.google.com/docs/cloud-messaging/concept-options#setting-the-priority-of-a-message). FCM high-priority messages let you reliably wake your app to access the network, even if the user's device is in Doze or the app is in App Standby mode. In Doze or App Standby mode, the system delivers the message and gives the app temporary access to network services and partial wakelocks, then returns the device or app to the idle state.

High-priority FCM messages do not otherwise affect Doze mode, and they don't affect the state of any other app. This means that your app can use them to communicate efficiently while minimizing battery impacts across the system and device.

As a general best practice, if your app requires downstream messaging, it should use FCM. If your server and client already uses FCM, make sure that your service uses high-priority messages for critical messages, since this will reliably wake apps even when the device is in Doze.

Support for Other Use Cases

Almost all apps should be able to support Doze by managing network connectivity, alarms, jobs, and syncs properly, and by using FCM high-priority messages. For a narrow set of use cases, this might not be sufficient. For such cases, the system provides a configurable whitelist of apps that are **partially exempt** from Doze and App Standby optimizations.

An app that is whitelisted can use the network and hold partial wake locks (https://developer.android.com/reference/android/os/PowerManager.html#PARTIAL_WAKE_LOCK) during Doze and App Standby. However, **other restrictions still apply** to the whitelisted app, just as they do to other apps. For example, the whitelisted app's jobs and syncs are deferred (on API level 23 and below), and its regular `AlarmManager` (https://developer.android.com/reference/android/app/AlarmManager.html) alarms do not fire. An app can check whether it is currently on the exemption whitelist by calling `isIgnoringBatteryOptimizations()` (https://developer.android.com/reference/android

/os/PowerManager.html#isIgnoringBatteryOptimizations(java.lang.String)).

Users can manually configure the whitelist in **Settings > Battery > Battery Optimization.** Alternatively, the system provides ways for apps to ask users to whitelist them.

- An app can fire the `ACTION_IGNORE_BATTERY_OPTIMIZATION_SETTINGS` (https://developer.android.com/reference/android/provider/Settings.html#ACTION_IGNORE_BATTERY_OPTIMIZATION_SETTINGS) intent to take the user directly to the **Battery Optimization**, where they can add the app.
- An app holding the `REQUEST_IGNORE_BATTERY_OPTIMIZATIONS` (https://developer.android.com/reference/android/Manifest.permission.html#REQUEST_IGNORE_BATTERY_OPTIMIZATIONS) permission can trigger a system dialog to let the user add the app to the whitelist directly, without going to settings. The app fires a `ACTION_REQUEST_IGNORE_BATTERY_OPTIMIZATIONS` (https://developer.android.com/reference/android/provider/Settings.html#ACTION_REQUEST_IGNORE_BATTERY_OPTIMIZATIONS) Intent to trigger the dialog.
- The user can manually remove apps from the whitelist as needed.

Before asking the user to add your app to the whitelist, make sure the app matches the acceptable use cases (#whitelisting-cases) for whitelisting.

> **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.

Testing with Doze and App Standby

To ensure a great experience for your users, you should test your app fully in Doze and App Standby.

Testing your app with Doze

You can test Doze mode by following these steps:

1. Configure a hardware device or virtual device with an Android 6.0 (API level 23) or higher system image.
2. Connect the device to your development machine and install your app.
3. Run your app and leave it active.
4. Force the system into idle mode by running the following command:

```
$ adb shell dumpsys deviceidle force-idle
```

5. Observe the behavior of your app after you reactivate the device. Make sure the app recovers gracefully when the device exits Doze.

Testing your app with App Standby

To test the App Standby mode with your app:

1. Configure a hardware device or virtual device with an Android 6.0 (API level 23) or higher system image.
2. Connect the device to your development machine and install your app.
3. Run your app and leave it active.
4. Force the app into App Standby mode by running the following commands:

```
$ adb shell dumpsys battery unplug
$ adb shell am set-inactive <packageName> true
```

5. Simulate waking your app using the following commands:

```
$ adb shell am set-inactive <packageName> false
$ adb shell am get-inactive <packageName>
```

6. Observe the behavior of your app after waking it. Make sure the app recovers gracefully from standby mode. In particular, you should check if your app's Notifications and background jobs continue to function as expected.

Acceptable Use Cases for Whitelisting

The table below highlights the acceptable use cases for requesting or being on the Battery Optimizations exceptions whitelist. In general, your app should not be on the whitelist unless Doze or App Standby break the core function of the app or there is a technical reason why your app cannot use FCM high-priority messages.

For more information, see Support for Other Use Cases (#support_for_other_use_cases).

Type	Use-case	Can use FCM?	Whitelisting acceptable?	Notes
Instant messaging, chat, or calling app.	Requires delivery of real-time messages to users while device is in Doze or app is in App Standby.	Yes, using FCM	Not Acceptable	Should use FCM high-priority messages to wake the app and access the network.
		Yes, but is not using FCM high-priority messages.		
Instant messaging, chat, or calling app; enterprise VOIP apps.		No, can't use FCM because of technical dependency on another messaging service or Doze and App Standby break the core function of the app.	Acceptable	
Task automation app	App's core function is scheduling automated actions, such as for instant messaging, voice calling, new photo management, or location actions.	If applicable.	Acceptable	
Peripheral device companion app	App's core function is maintaining a persistent connection with the peripheral device for the purpose of providing the peripheral device internet access.	If applicable.	Acceptable	
	App only needs to connect to a peripheral device periodically to sync, or only needs to connect to devices, such as wireless headphones, connected via standard Bluetooth profiles.	If applicable.	Not Acceptable	