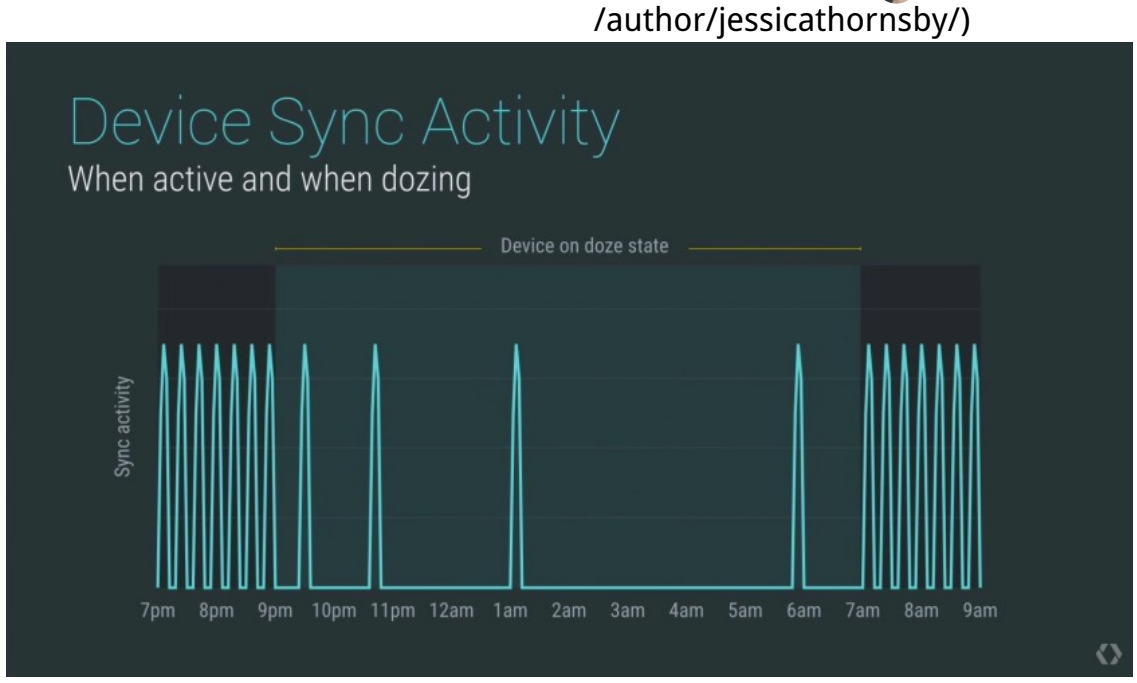
Get your apps ready to doze

ANDROID DEVELOPMENT (HTTP://WWW.ANDROIDAUTHORITY.COM/ANDROID-DEVELOPMENT/) by Jessica Thornsby (http://www.androidauthority.com/author/jessicathornsby/) September 9, 2016



Have you ever put your Android smartphone or tablet to one side, only to come back to it a few hours later and discover that it's burnt through *way* more battery power than you were expecting?

By default, Android devices receive information updates *constantly* – emails, social media messages, notifications from apps, syncing with your Google account and so on. So even if you don't interact with a device for an extended period of time, when you do eventually pick your smartphone or tablet up you'll find that it's bang up to date. However, there's a point where this convenience isn't worth the battery drain – no-one enjoys waking up in the morning to find their smartphone is now on 10% of battery because it spent the past 8 hours performing background work, while you were fast asleep.

Android 6.0 and higher attempts to strike a perfect balance between ensuring your smartphone or tablet is always relatively up to date (even if you haven't interacted with it for a while) without burning through unnecessary amounts of battery.

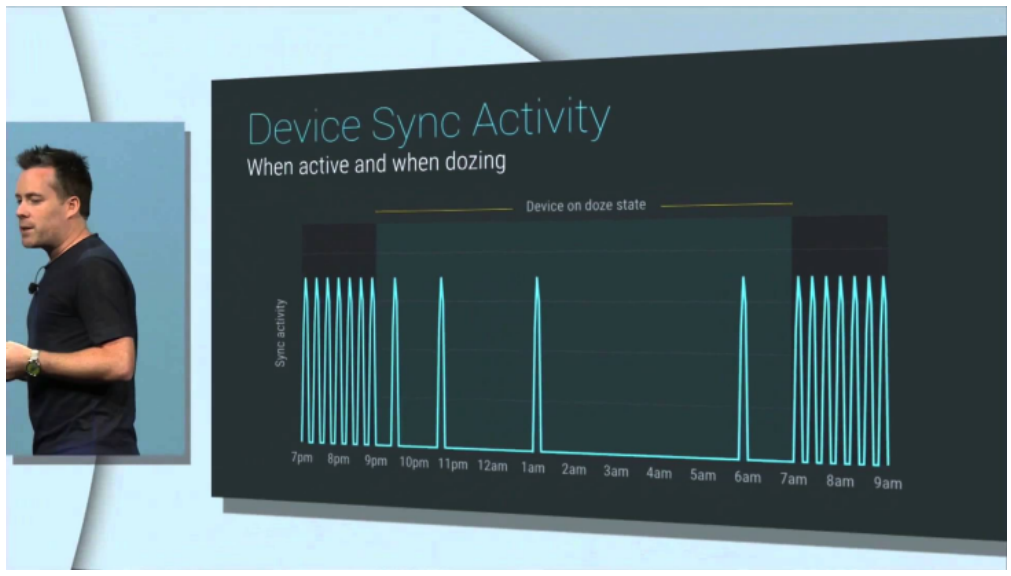
This new feature is known as Doze mode, and in this article we're going to look at how to update your apps, to make sure they place nicely with this new feature.

What is Doze Mode?

In the pre-Doze world, Android apps pretty much had free reign to perform whatever work they wanted in the background. While this was good for developers, who could create apps safe in the knowledge that said apps would be able to perform tasks whenever they needed (even if it meant waking an inactive smartphone or tablet) it wasn't such good news for the end-user who found themselves constantly needing to recharge their device.

Enter Doze.

When a device is unplugged, stationary, and the screen turned off, Doze mode will eventually kick in and put the the device into a sleep state – hence the name Doze, as the device is essentially taking a power nap.



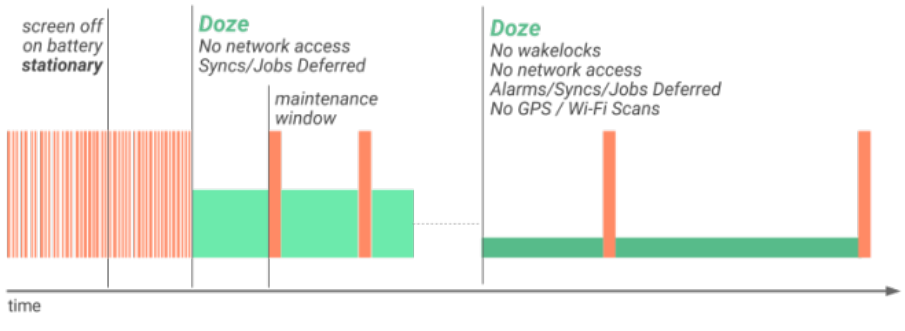
When a device is in Doze mode the system applies a range of battery-saving restrictions to all the apps on that device, as well as the device in general. For the duration of Doze mode, your app won't be able to

access the network, run sync adapters, fire standard alarms, run scheduled jobs, or acquire wakelocks. Think of Doze as an automatic flight mode – and we all know how much longer our battery lasts in flight mode!

As soon as a device no longer meets Doze's list of criteria (for example the user moves the device or connects a charger) the system will exit Doze and all apps can resume normal activity.

If an app does try to perform tasks during Doze mode, the system will group all these tasks and batch execute them as soon as the device exits Doze, or during a scheduled *maintenance window*.

Maintenance Windows



Imagine you put your Android smartphone or tablet down and don't touch it at *all* for a few hours (it's a stretch, I know). That device will eventually enter Doze mode, and from that point onwards it's pretty much in a state of suspended animation. When you do finally pick the device up again, all of your apps are at least a few hours out of date – not exactly a great user experience!

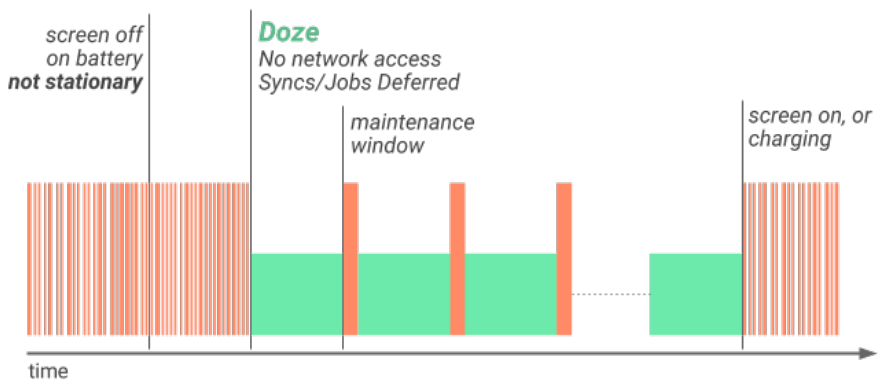
To ensure that Doze's battery savings don't come at the cost of the user experience, Android exits Doze for regularly scheduled maintenance windows. A device will resume normal operations during these windows, giving your app a chance to run all its deferred activities. At the end of each maintenance window, the device will re-enter Doze. When a device first enters Doze, these maintenance windows occur pretty frequently, although they do occur less frequently the longer a device is in Doze mode.

And this was pretty much all your needed to know about Doze mode and its maintenance windows – *until* Android 7.0 came along and added the disclaimer that a device didn't *necessarily* have to be stationary, in order to Doze.

Doze on the Go

When you think about it, an Android smartphone or tablet is rarely stationary. Your Android device probably spends a good chunk of its time in your pocket or bag, where it's going to get jostled around so much that it's unlikely to doze at all.

That's why Android 7.0 introduced 'Doze on the go,' a new tier of Doze mode that applies a subset of the regular, 'deep-Doze' restrictions when the device is running on battery power and the screen is turned off, but Doze is still detecting movement. This lightweight version of Doze ensures that users can benefit from Doze's battery saving features, even when they're on the go (hence the name!)



If a device's conditions change while it's dozing, that device may move between these two versions of Doze. So, if a device in Doze-light mode remains stationary for an extended period of time, then that device may sink into deep-Doze. At the other end of the scale, if a device in deep-Doze mode detects movement, *but* the screen remains off and the device is still unplugged, then it'll enter Doze-light mode, rather than exiting Doze completely.

The good news is that the recommended best practices are the same regardless of how deeply a device is dozing, so we can cover optimizing your app for both tiers of Doze, in one fell swoop.

Optimizing your Apps for Doze

By this point, you may be wondering how *any* app can provide a good user experience if it can't perform essential background work whenever it needs to. While it's true that Doze temporarily prevents applications from performing background activities, Doze is designed to have a minimal impact on your app's performance.

Maintenance windows crop up pretty frequently when a device first dips into Doze mode, and only start to occur less frequently when the device has been dozing for a while (the assumption is that the user has either left their device somewhere, or they've left it unplugged overnight and are actually fast asleep).

If your app has to wait a little longer in order to perform deferred work, then this isn't going to have a huge impact on the user experience – especially if the user is either nowhere near their device or it's the middle of the night and they're fast asleep.

However, there are some instances where you may need to make specific changes to your app, in order to provide a better Doze experience. In this section, I'll look at two features that Doze is *known* to interfere with, and the workarounds you'll need to use if your app includes these features. I'll also share one final trick you can resort to, just in case Doze completely breaks your app and you need a get-out clause from Doze's restrictions!

Receiving Messages in Doze Mode

If you're developing a messaging app, or an app that has some form of messaging functionality, then chances are your users aren't going to be too thrilled when your app doesn't notify them about important messages straight away, just because their device happened to be dozing when these messages were sent.

To make sure your app never fails to notify the user about an incoming message, you can use either Google Cloud Messaging (GCM) or Firebase Cloud Messaging (FCM). Both of these services have the power to push messages to a dozing device, as long as you mark those messages as high-priority.

> ***When your app is in Doze mode, standard AlarmManager alarms get deferred until the device enters its next maintenance window, or the device exits Doze completely.***

GCM and FCM attempt to deliver high-priority messages immediately. If your app receives a high-priority message during Doze, the system will wake the device and grant your app temporary network services and partial wakelocks so it can notify the user (just resist the temptation to use these temporary privileges as an excuse to perform work that really could have waited until the next maintenance window).

While it's easy to assume that *everything* your app does is important, waking a device from Doze mode will *always* have an impact on that device's battery, so you should only use this technique for messages that are truly time-critical.

Unless you have a good reason for marking a message as high priority, you should assume that all your message have the default priority. Messages marked as "normal" won't interrupt Doze mode, and will be delivered as soon as the device either enters a maintenance window or exits Doze completely.

Sounding the Alarm in Doze

Alarms are the other major feature you may need to adjust for Doze mode, so if you're developing an alarm app, or an application that has some form of alarm functionality, then this section is for you!

When your app is in Doze mode, standard AlarmManager alarms get deferred until the device enters its next maintenance window, or the device exits Doze completely. This presents a problem, as it's likely your users are going to *ooh* and *aww* over how little battery your app uses if they wind up getting into the office *hours* late because your app didn't sound their morning alarm when it was supposed to.

To create alarms that are immune to Doze, you'll need to use one of the following AlarmManager methods:

- setExactAndAllowWhileIdle. Use this method to create an alarm that executes in Doze mode at *exactly the time specified*.
- setAndAllowWhileIdle. Use this method if you need to be confident that an alarm will execute in Doze mode, but it's not crucial that this alarm fires at exactly the time specified. This may sound strange (surely the whole purpose of an alarm is that it goes off at a particular time?) but there's a few instances where you might want to use this method, rather than setExactAndAllowWhileIdle. For example maybe you're building an app that alerts the user to bank holidays and other important events, or an app that presents the user with a 'To Do' list at the start of each day. In these scenarios, is it really crucial for the alarm to fire at exactly the time specified?

Note, setAndAllowWhileIdle and setExactAndAllowWhileIdle are only available in Lollipop and higher.

Bear in mind that if your app wakes a device then it'll have an impact on that device's battery, so you should only use these new methods if the benefits outweigh the potential battery hit of waking a dozing device.

If you suspect an alarm can wait until the device exits Doze mode or enters a maintenance window, then you should use the standard set() and setExact() instead.

Requesting access to the whitelist

Doze shouldn't have a huge impact on most apps. Even if your app performs lots of background work then this work won't be ignored, it'll simply be deferred until the next maintenance window or until the device exits Doze (whichever comes first). And if you do need to make some explicit changes to your project in order to provide a better Doze experience, then most of the time this will be restricted to using GMC/FCM for time-sensitive messages, and using the new AlarmManager classes for important alarms.

However, occasionally Doze may break an app's core functionality, for example if you're developing a task automation app, then this app may hinge on being able to perform tasks when the user isn't interacting with their device. Alternatively, you may be developing a messaging app that can't use GCM or FCM for technical reasons.

If your app falls into either of these two very specific use cases, then you may need to request that the user adds your app to their 'whitelist,' at which point it'll be exempt from Doze's restrictions.

Users can build their own whitelist at any point, simply by opening their device's 'Settings' app, followed by 'Battery' and 'Battery Optimization,' finding the app(s) they want to add to their whitelist, and then setting that app's switch to 'Off.'

However if Doze mode breaks your app, then you should take a more proactive approach and explicitly request that the user adds your app to their whitelist. You have two options:

- Firing the ACTION_IGNORE_BATTERY_OPTIMIZATION_SETTINGS intent. This launches the device's 'Battery Optimization' screen, ready for the user to (hopefully) add your app to their whitelist.
- Adding the REQUEST_IGNORE_BATTERY_OPTIMIZATIONS permission to your project. This will trigger a system dialogue prompting the user to disable battery optimizations for your app, at which point your app will be exempt from Doze's restrictions.

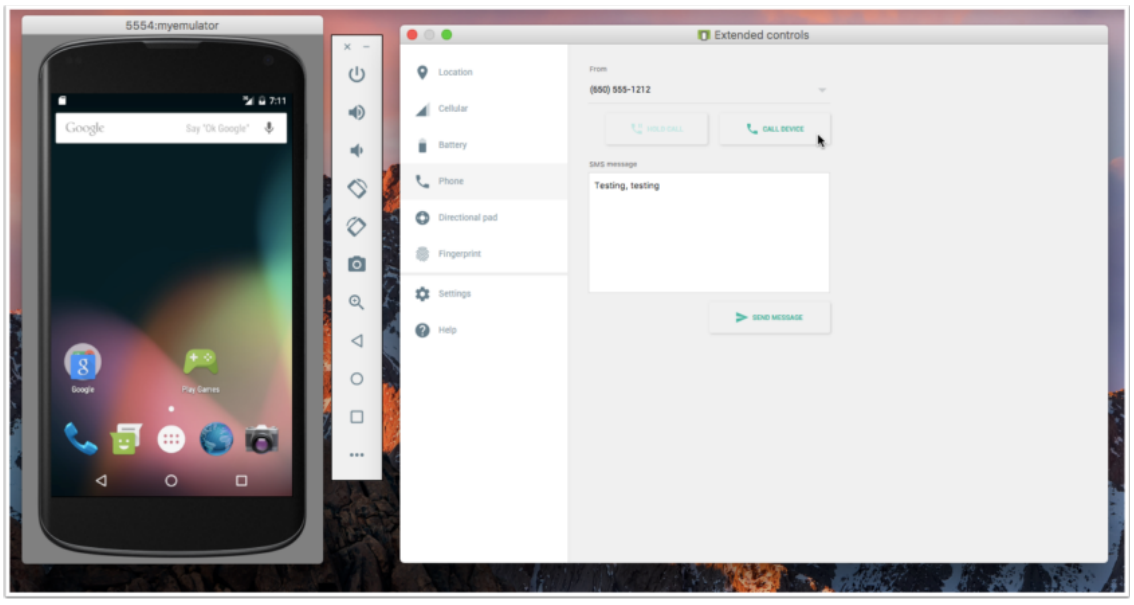
You can check whether your app has made it onto the user's whitelist at any point, by calling the isIgnoringBatteryOptimizations method.

Testing your app in Doze mode

The final step is testing how your app behaves in Doze, including ensuring that your app makes the most out of the mode's maintenance windows, and that your app recovers gracefully once the device exits Doze.

Rather than waiting for your device to slip into Doze mode naturally, you can cut to the chase and use adb commands to send a device into deep sleep in an instant.

The most effective way of testing your app's Doze performance, is to use an Android Virtual Device (AVD) that's running Android 6.0 or higher. You can then use the emulator tools to simulate different events that may occur while your app is subjected to Doze's restrictions, for example if you're developing a messaging app you should simulate your app receiving messages in Doze mode.



Make sure the app you want to test is installed on your AVD, then open a Terminal (Mac) or Command Prompt (Windows) and changing directory ('cd') so it's pointing at your Android SDK's 'platform-tool's folder, for example:

cd /Users//Library/Android/sdk/platform-tools

Make sure the app you want to test is running, then turn the AVD's screen off and simulate the device entering Doze mode by running the following adb commands:

adb shell dumpsys battery unplug

This tells the AVD to assume it's been unplugged from a power source.

adb shell dumpsys deviceidle step

This command takes the device through the various states it needs to sink through, before entering full-blown Doze. The Terminal will print the device's state each step of the way, so keep re-entering this command until the Terminal/Command Prompt window returns the Idle state.

Once your app is in Doze mode, spend some time testing how your app handles Doze in general, being on the lookout for anything that isn't working as you intended, or parts of your app that you could tweak in order to provide a better overall Doze experience.

In particular, make sure you simulate all the events that you suspect Doze might impact, for example if you want your SMS app to wake the device whenever it receives a new message, then simulate an incoming message and check that your app behaves as expected.

You should also check how your app handles the device leaving Doze mode; the easiest way is by turning the AVD's screen on and observing your app's behavior.

By default, adb's *deviceidle step* command glosses over the light-Doze phase and sends the device directly into a deep Doze, but you'll want to test that your app provides a good user experience in both Doze states.

To place an AVD into Doze-light mode, enter the following adb command:

$ adb shell dumpsys deviceidle step [light]

Wrapping Up

Do you have anymore tips for creating apps that play nicely with Android's Doze mode? Share them in the comments below!

ANDROID DEVELOPMENT (HTTP://WWW.ANDROIDAUTHORITY.COM/ANDROID-DEVELOPMENT/)

Battery (http://www.androidauthority.com/tag/battery/), Doze Mode (http://www.androidauthority.com/tag/doze-mode/)
Battery (http://www.androidauthority.com/tag/battery/), Doze Mode (http://www.androidauthority.com/tag/doze-mode/)

Jessica Thornsby (http://www.androidauthority.com/author/jessicathornsby/)

How to flash Android Oreo on your Nexus or Pixel

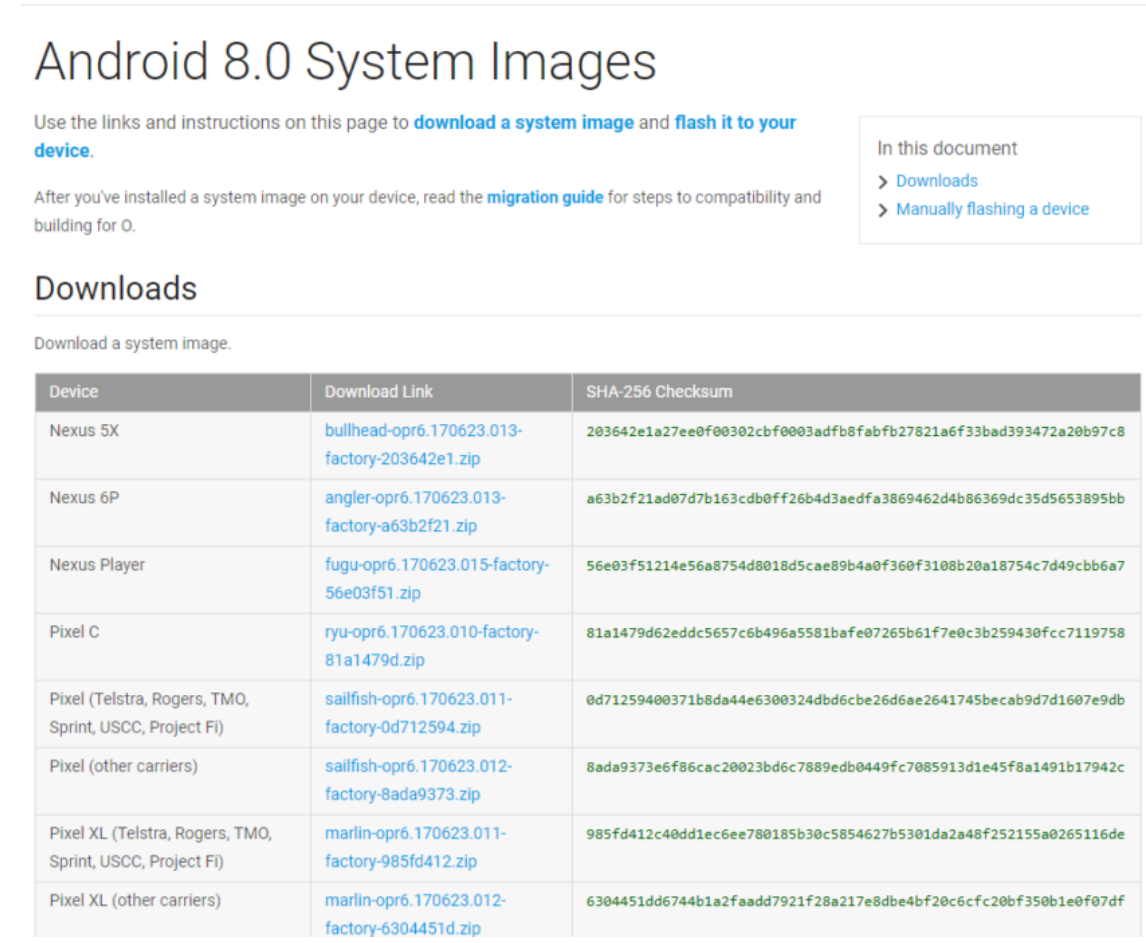
ANDROID DEVELOPMENT (HTTP://WWW.ANDROIDAUTHORITY.COM/ANDROID-DEVELOPMENT/) NEWS (HTTP://WWW.ANDROIDAUTHORITY.COM/NEWS/) by Jean-Luc Picard (http://www.androidauthority.com/author/teamaa/)

2 days ago

At today's **big Eclipse-centered event, (http://www.androidauthority.com/android-eclipse-event-794452/)** Google formally took the veil off of Android 8.0 Oreo. After several preview builds, the final, stable version is now preparing to land. While it shouldn't take very long for the update to formally hit select Nexus and Pixel devices, but for those that don't like waiting, you can manually flash it to a supported device.

So what devices will work with Oreo? The Nexus 5X, Nexus 6P, Pixel C, Pixel, Pixel XL, and Nexus Player all made the cut. As for how hard it is to flash? Actually, it's pretty easy. There are a few hoops to go through, but nothing too difficult. Of course, as with everything flashing, you're doing so at your own risk.

For those that are familiar with flashing, there's really nothing new about this process with Android Oreo. If you're never flashed an update at all, this guide will take you through everything you need to know about the process.



Android 8.0 System Images

Use the links and instructions on this page to **download a system image** and **flash it to your device**.

After you've installed a system image on your device, read the **migration guide** for steps to compatibility and building for O.

In this document
- Downloads
- Manually flashing a device

Downloads

Download a system image.

Device	Download Link	SHA-256 Checksum
Nexus 5X	bullhead-opr6.170623.013-factory-203642e1.zip	203642e1a27ee0f00302cbf0003adfb8fabfb27821a6f33bad393472a20b97c8
Nexus 6P	angler-opr6.170623.013-factory-a63b2f21.zip	a63b2f21ad07d7b163cdb0ff26b4d3aedfa3869462d4b86369dc35d5653895bb
Nexus Player	fugu-opr6.170623.015-factory-56e03f51.zip	56e03f51214e56a8754d8018d5cae89b4a0f360f3108b20a18754c7d49cbb6a7
Pixel C	ryu-opr6.170623.010-factory-81a1479d.zip	81a1479d62eddc5657c6b496a5581bafe07265b61f7e0c3b259430fcc7119758
Pixel (Telstra, Rogers, TMO, Sprint, USCC, Project Fi)	sailfish-opr6.170623.011-factory-0d712594.zip	0d71259400371b8da44e6300324dbd6cbe26d6ae2641745becab9d7d1607e9db
Pixel (other carriers)	sailfish-opr6.170623.012-factory-8ada9373.zip	8ada9373e6f86cac20023bd6c7889edb0449fc7085913d1e45f8a1491b17942c
Pixel XL (Telstra, Rogers, TMO, Sprint, USCC, Project Fi)	marlin-opr6.170623.011-factory-985fd412.zip	985fd412c40dd1ec6ee780185b30c5854627b5301da2a48f252155a0265116de
Pixel XL (other carriers)	marlin-opr6.170623.012-factory-6304451d.zip	6304451dd6744b1a2faadd7921f28a217e8dbe4bf20c6cfc20bf350b1e0f07df

What will you need before you get started?

- A compatible Nexus or Pixel device along with a USB cable to connect it to your computer.
- The Android SDK installed on your machine with ADB and Fastboot command successfully working. **Here's a tutorial on how to do that (https://developer.android.com/sdk/installing/index.html)**.
- You'll also need the appropriate factory image for your device. Go to **this website to download them (https://developer.android.com/about/versions/o/download.html#flashable-images)** and make sure you get the right one for your device. It's worth it to spend a moment to make sure you have the right one rather than have to deal with the issues of downloading the wrong one.
- You'll also need 7zip or a similar program that can handle .tgz and .tar files. You can **download 7zip for free here (http://www.7-zip.org/)**.
- You will also need to unlock your bootloader. Beware, this will erase your data. It's also worth noting that flashing a factory image will also erase your data. Make sure to back it up!

Unlocking your bootloader

Before you get started, if your phone's bootloader has never been unlocked, you'll need to do a few extra steps before manually installing Android O. Remember, opening the bootloader will reset your phone, losing all personal data.

With that out of the way:

1. If you haven't already, you'll need to turn on developer options. To do so, you just need to go to "About Phone" and tap seven times on "Build Number".

2. From there, enable USB debugging and OEM unlock on your Nexus/Pixel device. These can be found in "developer options" section.

3. Go ahead and plug in your device to your PC via USB cable now.

4. Open a command window on the PC.

5. Boot your Pixel device into bootloader mode using the following command: *adb reboot bootloader (if it requests you to authorize this, say yes)*

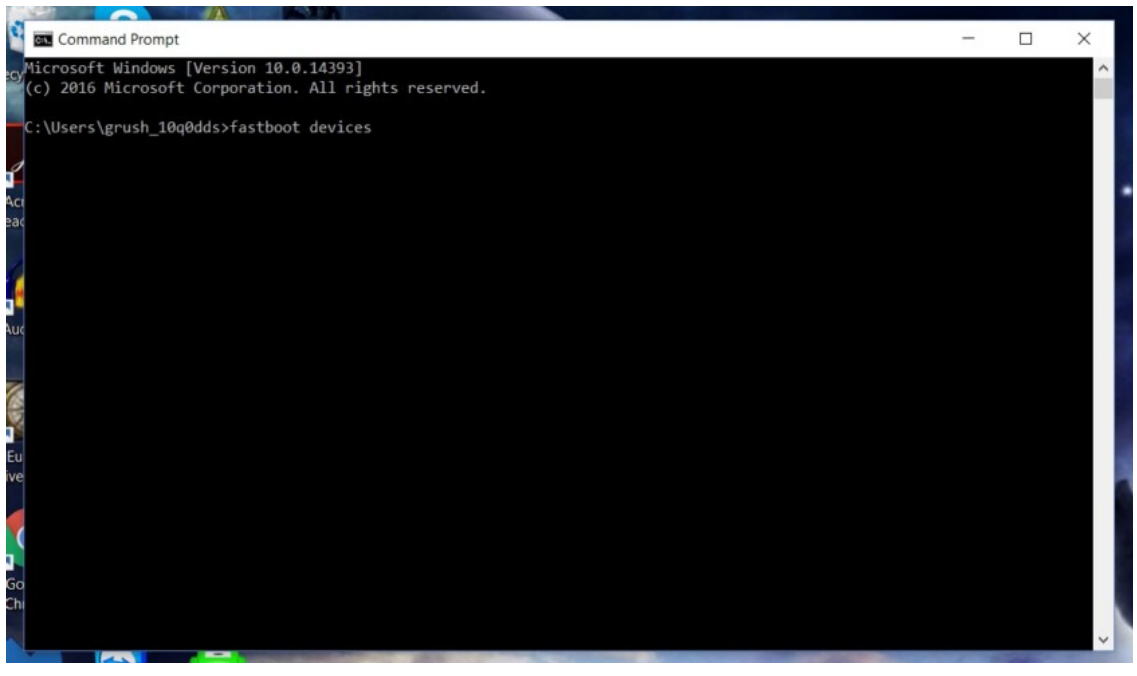
6. Your device will boot into bootloader mode. From here type the command: *fastboot flashing unlock*

7. For the Pixel family, you'll get a confirmation screen. Press Volume Up to highlight yes, and power to select it. This will begin bootloader unlocking process.

8. Once unlocked, your device will reboot into bootloader mode. Now you simply need to type *fastboot reboot.*

9. During this reboot, your device will go through a factor reset. This part is now over.

How to manually install Android Oreo on a Nexus or Pixel device



Keep in mind that this process is pretty straightforward, but things can and do go wrong if you don't carefully follow instructions. With that out of the way, here's what you need to do:

1. If you aren't in the bootloader menu still, you'll need to go back in. From here, you'll want to test that your device and PC are communicating by typing *fastboot devices* — if it comes back with your device's serial number, you're golden. If not, you'll probably need to hit up Google search for some troubleshooting.
2. Next, it's time to prepare the factory image you downloaded earlier. On your computer, use 7zip to extract the .tgz file you downloaded. Use 7zip a second time to extract the .tar file you extracted from the .tgz. When you're done, you should have a folder with several files in it.
3. Copy all of these files and paste them in the platform-tools folder in the Android SDK on your computer. If you followed the above tutorial, this should be under the C drive, then under Program Files (x86) on Windows. Linux users, you know where you put it.
4. There are two flash-all files. If you're in Windows, you'll want to double click the one that has the gear logo and says "Windows Batch File" on the right. If you're on Linux, you'll want to double click the flash-all.sh.
5. At this point a box should pop up and you should see the installation taking place. While this is going on, do not unplug your device for any reason. Let it do its thing.
6. Once the installation process as finished, your device will automatically reboot and you should see the Android O boot animation start up. You can now safely disconnect your device from your computer.

That should be it. If this method doesn't work for you for whatever reason, there are a few other ways to go about things and to figure out the method that works best for you, we suggest either ask away in the comments to see if anyone can help you out, or hit up Google search. Additionally, checkout the **Android Authority Forums (http://forums.androidauthority.com/)** and ask there if you can't quite get things figured out. Good luck, have fun, and we hope you enjoy the latest preview!

ANDROID DEVELOPMENT (HTTP://WWW.ANDROIDAUTHORITY.COM/ANDROID-DEVELOPMENT/)

NEWS (HTTP://WWW.ANDROIDAUTHORITY.COM/NEWS/)

Android O (http://www.androidauthority.com/tag/android-o/)
Android O (http://www.androidauthority.com/tag/android-o/)

Team AA (http://www.androidauthority.com/author/teamaa/)

Android Instant Apps: what do they mean for users and developers?

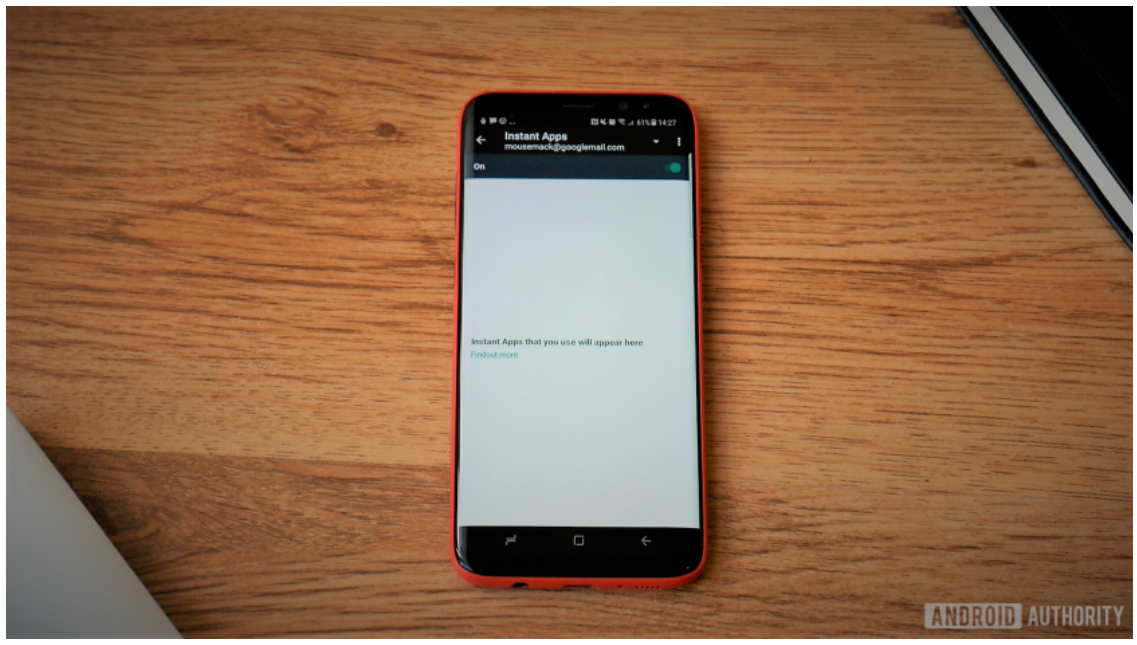
ANDROID DEVELOPMENT (HTTP://WWW.ANDROIDAUTHORITY.COM/ANDROID-DEVELOPMENT/) FEATURES (HTTP://WWW.ANDROIDAUTHORITY.COM/FEATURES/)

by **Adam Sinicki** 6 days ago (http://www.androidauthority.com/author/adamsinicki/)

Do you really need to download an app to own it?

I'm not being philosophical here (do we truly own anything?) but when you consider how quickly and easily you can download and install something like a flashlight app, it makes you wonder what the advantage to *keeping* it on your device actually is. Unless you work down a mine, it's unlikely you'll need those special use case apps all that often; so as long as you're able to access an app quickly when you need it most, what benefit is there to having it take up space the rest of the time?

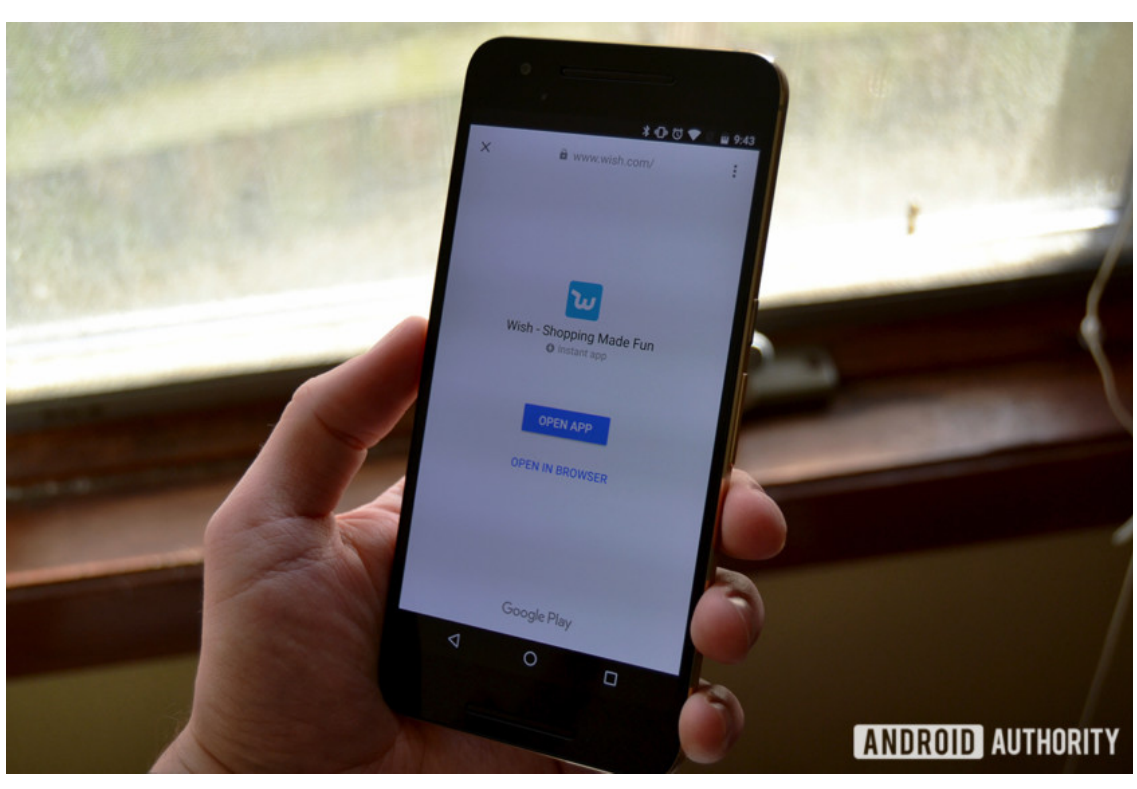
Android Instant Apps are Google's answer to this dilemma. Instant Apps is a feature that lets you use an app *without* needing to fully download it onto your phone: just find it in the Play Store and click 'Open App'. Better yet, it allows you to jump to a specific activity within an app you don't have installed, simply by tapping a URL. It's already available for some users and is in the process of rolling out to the rest of us. But what does it really mean for you? And how should developers adapt to this new feature?



Using Android instant apps

To access Android instant apps on your device, you simply need to turn the option on in your settings menu – as long as you have a Nexus or Pixel device. Head to *Personal* > *Google* and then *Services*. Now just toggle *Instant Apps* on and then tap 'Yes, I'm In' when the notice comes up. You can try it out with some of the early adopters like BuzzFeed or Wish. If you don't have one of those devices, then you may still be able to find it in your settings, but the apps won't be available yet for you to try. Don't worry though, they're coming!

Android Instant Apps can also be launched from a URL. This is similarly handy because it essentially extends the web browsing experience to add more power and native functionality – not to mention allowing us to share experiences from within apps.



What this could mean for users

So, what does this mean for users? Should you be excited, worried or indifferent?

On the whole, this is pretty exciting news and many expect it to be a game changer for the way we use our devices. There are many apps in the Play Store that are typically 'one-use-only' affairs, or things you will seldom use again. And there are still plenty of us with a paltry 16 GB of internal storage on our devices (or even less).

> ***Most of us would prefer to use a native app over a website where possible but we don't want the hassle of installing it.***

While it's not too much trouble right now to install an app and then uninstall it immediately after, Android Instant Apps promises to streamline the process even further so that you can save time and enjoy even greater efficiency from your device. Most of us would prefer to use a native app over a website where possible but we don't want the hassle of installing it.

This is what you call 'having your cake and eating it too'.

Better yet, is the promise of being instantly dropped into a specific useful page in an app with no hassle involved in installing it. The example Google gave at the recent developer conference was that a user could tap a parking meter with their phone to instantly open up a parking app (through NFC) on the payment page, ready to pay with Android Pay.

Another example was that you might be able to share a crossword puzzle with a friend over WhatsApp. Had particular fun with that puzzle? Then you can send the link over and the recipient will be able to dive right to that page within the app, with no need to install it first or even navigate through the menu.

Web browsing will become far more seamless, as sites switch between web pages, apps and back again. In future, we might even see sites using links to launch instant apps from other developers. While checking out nearby restaurants in Maps you could click a review link in Yelp and then open up the Uber app to book a taxi!

> ***Web browsing will become far more seamless, as sites switch between web pages, apps and back again.***

Speaking of the future, it is definitely easy to look at Android instant apps as being a big step toward an inevitable evolution for our online experiences. With data plans becoming more and more generous, connections becoming increasingly speedy and cloud storage being commonplace; it's only a matter of time until we no longer have to download *anything*. Instant apps still actually download and install a portion of the software, but in future even the processing is likely to be outsourced to a server somewhere and that will drastically reduce the need for expensive hardware.

This is a small step in that direction, but it is a positive one.

Security and limitations

The worry that some might have reading this, is that it could present security issues. What if a webpage were to temporarily install an app on your phone that could bill you through Android Pay for instance?

> ***An app can't start billing you or reading your contacts unless you say that it can.***

While Android instant apps may introduce some new security concerns, there are measures in place that ensure users shouldn't need to worry for the most part. All network traffic from inside the apps will use HTTPS. Signing in will need to be handled by **Smart Lock (https://developers.google.com/identity/smartlock-passwords/android/)** (which also keeps the process nice and speedy) and users will need to give permission just as they do for installed apps. An app can't start billing you or reading your contacts unless you say that it can.

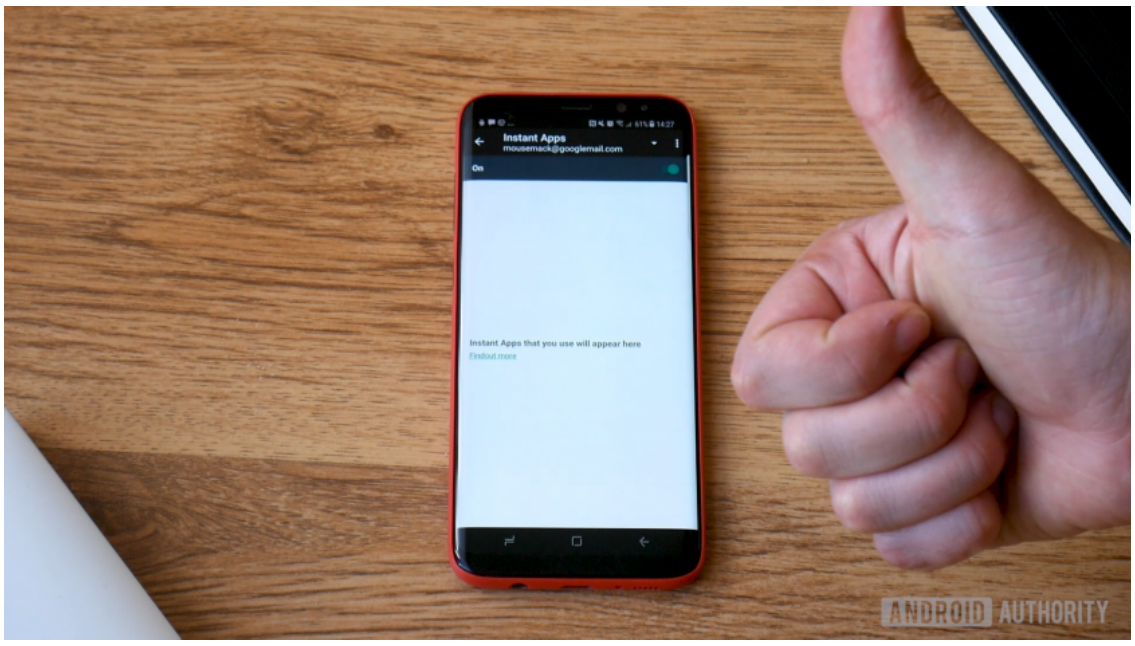
Google's Instant App FAQs page, tells us that these apps can use the following permissions:

- BILLING
- ACCESS_COARSE_LOCATION
- ACCESS_FINE_LOCATION
- ACCESS_NETWORK_STATE
- CAMERA
- INSTANT_APP_FOREGROUND_SERVICE only in Android O.
- INTERNET
- READ_PHONE_NUMBERS only in Android O.
- RECORD_AUDIO
- VIBRATE

Anything not in this list is not supported by Instant Apps. Notice that things like Bluetooth, set alarm, use fingerprint and set wallpaper are missing.

Other limitations include the lack of support for background services (apps that run potentially without the

user's knowledge), for push notifications, for accessing external storage, or for looking at installed apps on a device. Instant apps also won't be able to change settings on the user's device, such as their wallpaper.



As you might expect, there is also a file size limit for instant app downloads too, that being 4 MB for each 'feature' or each page (think activity) of an app. This of course creates more potential limitations. It means, for example, that developers can't pack an app full of rich media, though of course there's nothing to stop them from streaming media from elsewhere.

But it does pretty much discount something like a fully 3D game. At the moment at least. Google has this to say on the subject:

> *"Games are a highly specialized category of apps, and often have unique tools, large assets, and high performance requirements. Even so, we are interested in exploring game use cases. Check the* ***Android Instant Apps posts on StackOverflow (http://stackoverflow.com/questions/tagged/android-instant-apps)***"

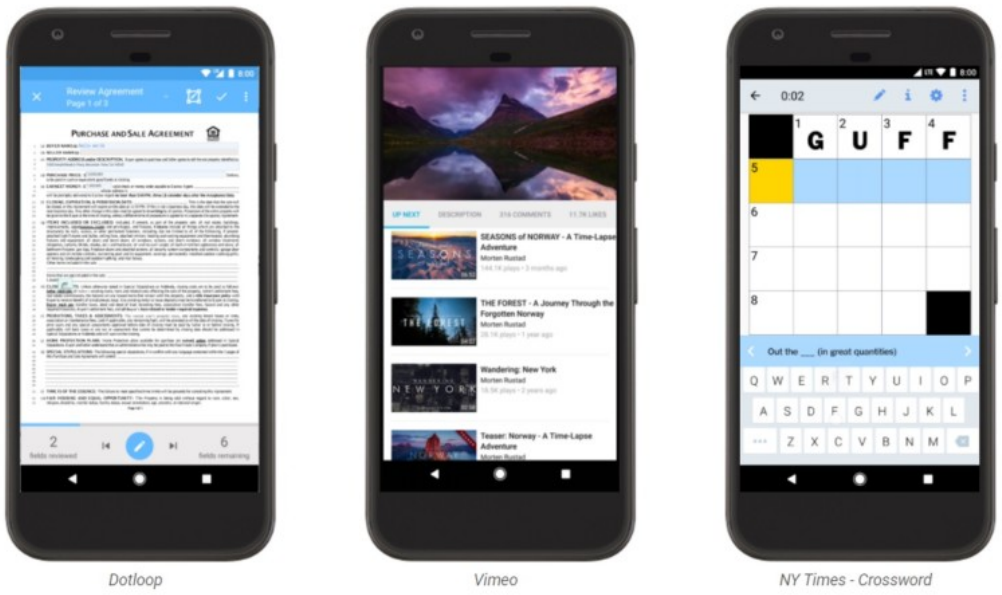
In the short term, there is nothing to stop you from creating games, but they will generally be things like puzzle games or *very* basic 2D platformers at a push. Hardware acceleration with OpenGL ES 2.0 is supported however, so future potential is there.

> ***Time will tell whether some of these restrictions are lifted or whether more are introduced.***

Time will tell whether some of these restrictions are lifted or whether more are introduced. It will also be interesting to see how developers and brands adopt the feature. One thing worth bearing in mind is the fact that iOS does not currently have a comparable service. Businesses might be cautious about introducing new experiences that only a certain section of their audience will be able to appreciate – but again, only time will tell.

What this means for businesses and developers

Businesses should be excited at the prospect of Android instant apps though, as this creates a lot of new opportunities for increased engagement and probably *sales* as a result. As we've already touched on, Android Instant Apps will provide a way for websites to link to more dynamic content for mobile users and this in turn will allow for the use of location awareness, in-app purchases and more. The real appeal for a business then, is the ability to let a user seamlessly order a pizza or buy a product through their app, or to get directions to a store. And the ability to share links to pages within apps will greatly increase the discoverability of those apps and potentially lead to more traffic. Those users that don't like your app are also much less likely to leave a negative review too.



Dotloop *Vimeo* *NY Times - Crossword*

Instant apps will be a must for businesses that want to leverage the maximum marketing potential of their mobile apps then. However, for developers that make their living from app installs or from advertising, the benefits may be less clear cut. If you make your money from ads, then you may benefit from having more users frequenting your individual activities (and Firebase is supported). On the other hand though, the lack of requirement to install the app, might reduce how often some users come *back* to your app after their first encounter.

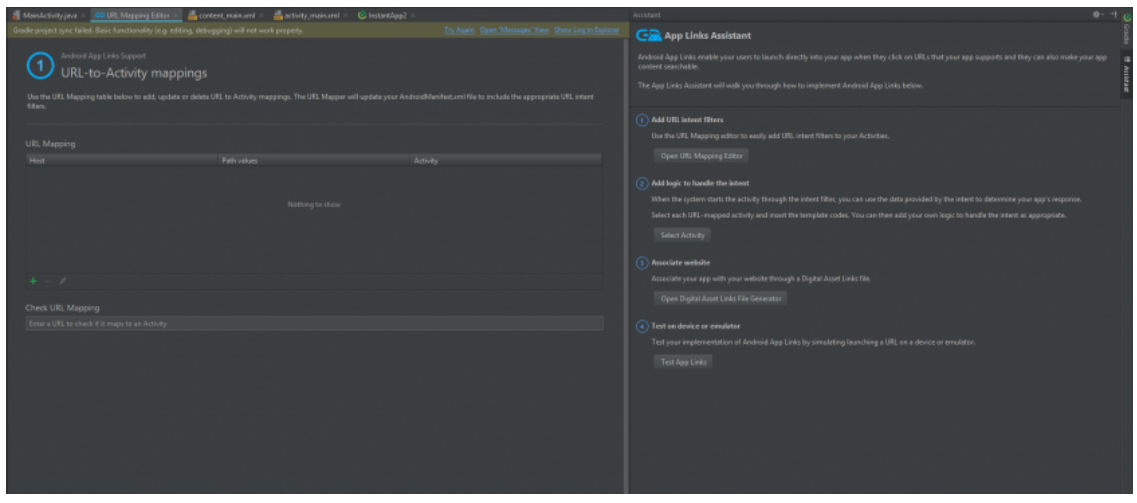
> ***Instant apps will be a must for businesses that want to leverage the maximum marketing potential of their mobile apps.***

Instant apps will be supported by Android versions dating all the way back to Jelly Bean, meaning that they will be available to millions of users but seeing as they *won't* be available on iOS, some web pages might be reluctant to make them a big part of their strategy.

How developers can implement Android instant apps

An in-depth tutorial is beyond the scope of this post but we can quickly go over what is involved in creating an instant app.

The good news is that Android Studio 3.0 will come with instant app support out of the box. You'll download the Android Instant Apps SDK from the SDK Manager and then you'll use the App Links Assistant for easily adding your links. The emulators will now support testing on a local environment too (deep links were previously tested using ADB).



(http://cdn03.androidauthority.net/wp-content/uploads/2017/08/Instant-Apps-App-Links-Assistant-Android-Studio.png)

The biggest difference is the use of a new type of construct: feature modules. These work like libraries with their own code, resources and manifests and will be accessed in the same way from your installable app, but they will build as individual .apks for your instant apps. An instant app module will act like a container (a .zip) for your feature modules.

So to convert a regular app into an instant app, you'll first use the App Links Assistant to modify your manifest and define entry points and URLs to access them. This works in a similar manner to the way that you would currently insert a deep link for linking directly to activities within a pre-installed app on your device.

You'll then convert your application module and place it within a base feature module. You'll rename the application to be a feature and change the Gradle file so that instead of com.android.application, you have com.android.feature. You'll also add a line to Gradle to define your base feature. You'll then add an application module for your current app, a 'base' feature module for the main app and a feature module for each instant app. All your app modules will build off the base feature module and so will have dependencies added to the Gradle files. There are a few additional steps and you'll find a more detailed

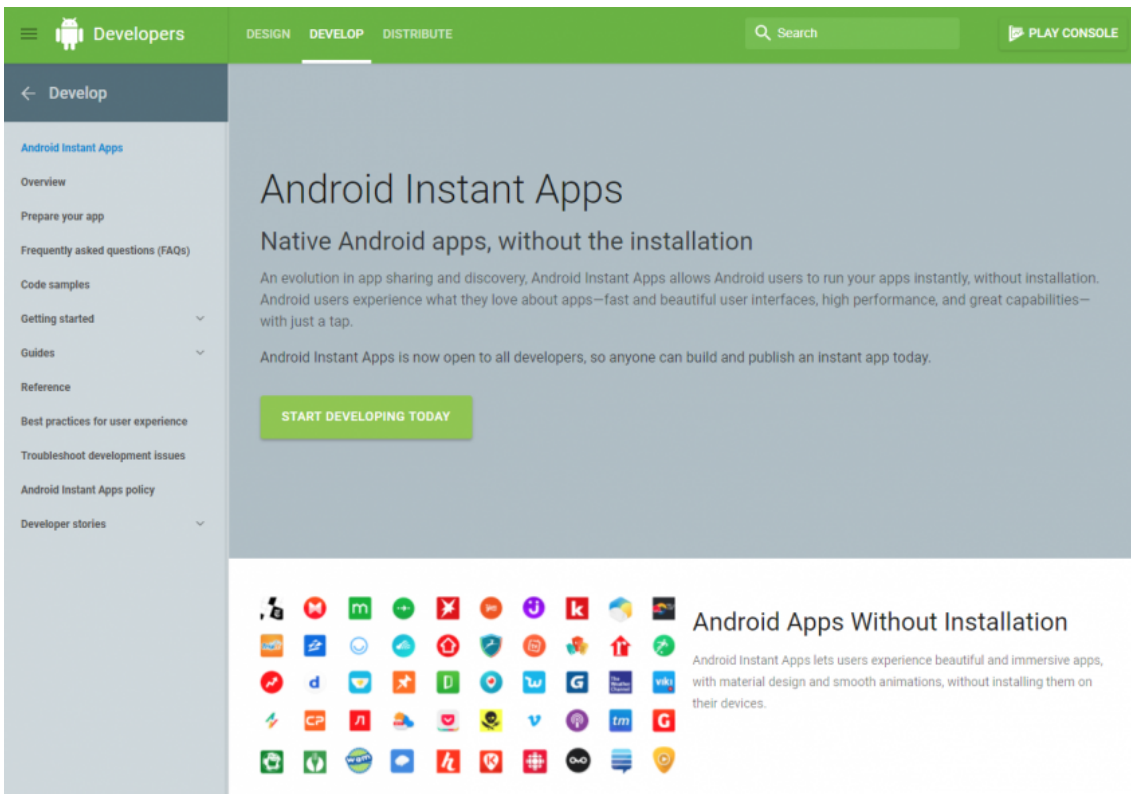
explanation below.

Google assures us that this whole process can be handled in less than a day for a basic app, but it will depend a lot on your current app structure, as well as the scale of the project you have in mind. If you want to create a store app and make every product listing its own feature, then you're going to have numerous additional steps compared to building a regular app – but for larger projects it could take significantly longer. It certainly does introduce a number of additional steps if you want to make each page in a store run as a separate instant app for instance.

Best practices

Android instant apps introduce a host of new challenges for designers and developers and will require a new design language and way of thinking.

Google has **shared some best practices (https://developer.android.com/topic/instant-apps/ux-best-practices.html#dont_branch_your_ui)** here. For example, developers *mustn't* aggressively urge users to download the full application. Developers can use an install button to prompt this but must do so in a subtle manner. Prompts should be limited to no more than two or three instances. Likewise, they need to avoid branching their UI and they are definitely encouraged not to add splash screens to individual pages. Smart Lock should be used for identity to avoid users having to continually log in and out of the apps and sites.



It's definitely worth reading through the full guide, but the best way to summarise this is to keep the transition between the web page and the app as seamless as possible while *also* remembering that users will be loading these pages from within the regular app.

Conclusion

So what do you make of Android instant apps? Can you see yourself using them? Developers: will you be converting your current apps, or using this feature for future projects?

Personally I see a lot of appeal and hope this is a step toward a 'no downloads future'. For now, the ability to link friends directly to pages within apps will hopefully increase engagement and introduce a range of new use cases.

Success will likely hinge on the willingness of developers to put in that extra time though, which in turn will depend on just how ready users are to change their relationship with their software.

ANDROID DEVELOPMENT (HTTP://WWW.ANDROIDAUTHORITY.COM/ANDROID-DEVELOPMENT/)

FEATURES (HTTP://WWW.ANDROIDAUTHORITY.COM/FEATURES/)

Android Development (http://www.androidauthority.com/tag/android-development/), Android Developer Tools (http://www.androidauthority.com/tag/android-developer-tools/), Android Instant Apps (http://www.androidauthority.com/tag/android-instant-apps/)
Android Developer Tools (http://www.androidauthority.com/tag/android-developer-tools/), Android Development (http://www.androidauthority.com/tag/android-development/), Android Instant Apps (http://www.androidauthority.com/tag/android-instant-apps/)

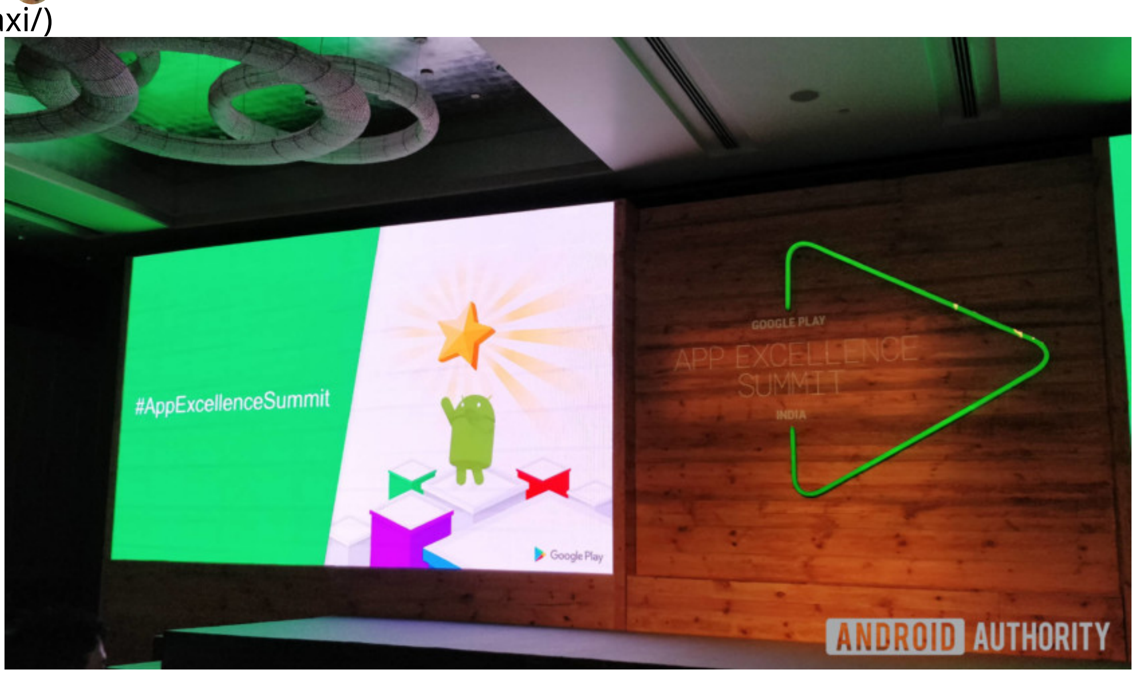
Adam Sinicki (http://www.androidauthority.com/author/adamsinicki/)

Google announces 'Made for India' initiative to showcase apps optimized for Indian market

ANDROID DEVELOPMENT (HTTP://WWW.ANDROIDAUTHORITY.COM/ANDROID-DEVELOPMENT/) **INDIA (HTTP://WWW.ANDROIDAUTHORITY.COM/INDIA/)**

NEWS (HTTP://WWW.ANDROIDAUTHORITY.COM/NEWS/)

ww.androidauthority.com by Abhishek Baxi 3 weeks ago
bhishekbaxi/)



At the first ever Google Play App Excellence Summit in India, Google today announced its new 'Made for India' initiative.

As part of the initiative, Indian developers can apply for a chance to have apps that are specifically optimized for the Indian market to be showcased on the Google Play Store in India in a special section.

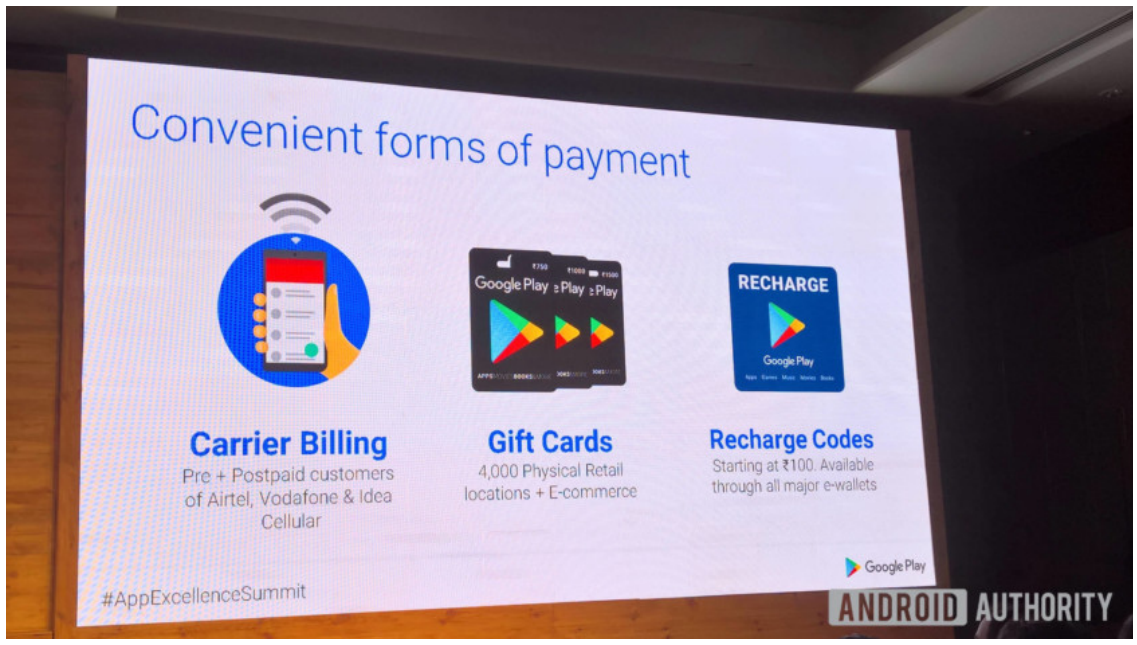
Made for India is an initiative to discover and showcase developers who are building high quality apps and optimizing them for Indian users. All you have to do is to apply at **g.co/play/madeforindia (http://g.co/play/madeforindia)**.

Specifically, the company is looking for factors like innovation, minimized data consumption, optimized battery consumption, optimized for internet connectivity, useful offline state, device compatibility for apps, app size reduction, and localization support.

The reason for Google's interest in the market is obvious. More than 70% of internet users in India go online primarily via their smartphones and this number is growing at a phenomenal rate. Interestingly, there are now more people using Android devices in India than in the US.

During her note to kick off the summit earlier today, Purnima Kochikar, Director – Business Development, Google Play, shared that people in India install more than a billion apps every month from Google Play. And while this is a staggering number already, the number of apps installed in India has grown by 150% each year.

Since Google has introduced diverse and convenient forms of payment for Indian consumers, the consumer spend on apps and games is growing at 3X in India.



At the summit, Google brought together over 700 Indian app and games developers, where the company shared tips and tools to help developers create the best quality Android apps that are locally relevant.

Last year, Google announced 'Build for Billions' guidelines to help developers overcome challenges such as varying network connectivity, device specifications, and high data costs. The playbook, **available on Google Play (https://play.google.com/store/books/details?id=cJEjDAAAQBAJ)**, is a step-by-step guide for developers to learn about the features, tools, and best practices to get apps ready to meet the needs of billions of Android users in India as well as globally.

With initiatives like 'Build for Billions' and 'Make for India', Google aims to help Indian developers of all sizes build successful, locally relevant businesses.

ANDROID DEVELOPMENT (HTTP://WWW.ANDROIDAUTHORITY.COM/ANDROID-DEVELOPMENT/)

INDIA (HTTP://WWW.ANDROIDAUTHORITY.COM/INDIA/) NEWS (HTTP://WWW.ANDROIDAUTHORITY.COM/NEWS/)

Google (http://www.androidauthority.com/tag/google/), Google Play Store (http://www.androidauthority.com/tag/google-play-store/), Google Play (http://www.androidauthority.com/tag/google-play/), App Excellence Summit (http://www.androidauthority.com/tag/app-excellence-summit/)
App Excellence Summit (http://www.androidauthority.com/tag/app-excellence-summit/), Google (http://www.androidauthority.com/tag/google/), Google Play (http://www.androidauthority.com/tag/google-play/), Google Play Store (http://www.androidauthority.com/tag/google-play-store/)

Abhishek Baxi (http://www.androidauthority.com/author/abhishekbaxi/)
A technology columnist and a digital consultant, he quit Microsoft in 2011 to go independent and write more, watch a lot of movies, and travel randomly.