

Wakelocks for Android\*

Christopher Bird

# Android Power Management – Introducing Wakelocks

Most people have probably dealt with the frustration of trying to use a cell phone with a short battery life. It stinks. A cell phone with a dead battery is about as useful as a cement brick. Consumers generally get very upset if their phone’s battery doesn’t last throughout the day, considering it a major inconvenience to be without a phone while it charges.

New and improved software allows tasks traditionally requiring a laptop or PC to now be done while away from any desk using a smartphone. However, the small form factor of smartphones severely limits the allowed battery size compared to a laptop. It’s a tricky requirement to have a phone imitate the functionality of a laptop, while having the strict requirement of retaining all-day battery life.

Android and other mobile OSs have achieved longer battery life by implementing an aggressive power manager. Shortly after using a phone, its display turns off and the CPU goes into a deep power state, thus sipping as little power as possible when not in use. This is how phones, when they are not used, are able to last several days on a single charge. With Android’s power manager, the normal assumption is that when the display is off, so is the CPU.

However, Android developers have the freedom (and mighty power) to prevent an Android device from going to sleep. They may wish to keep the CPU active, even when the display is off. Or perhaps they want to be able to prevent the display from automatically turning off during a specific activity. For this purpose, Google\* has included wakelocks into their PowerManager APIs[[1]](#endnote-1). Applications that want to prevent a device from going to sleep can grab a wakelock. As long as there is an active wakelock on the system, the device cannot enter suspend until the wakelock is released. When using wakelocks, it’s extremely important to understand that you must properly release wakelocks when they’re not needed because an unreleased wakelock will quickly drain a device’s battery, by not entering its default state to conserve energy.

To see when a developer might use a wakelock, this article will show some of the default Android applications (and usage models) that use them in Android 4.0. Then the article will introduce a sample application, “Wakelocks,” from the SDPSamples collection to show how wakelocks are done in code.

# App Usages of Wakelocks

With Android, there is a way to see which services are holding the wakelocks that are preventing the system from going into low-power states. The file /proc/wakelocks on the device shows a list of defined services and drivers that use wakelocks. By monitoring the contents of the /sys/power/wake\_lock file (root access required), you can know when there’s a lock on the CPU resources, and what service is holding the wakelock2. I was able to capture a few use cases in which wakelocks were used on my Galaxy Nexus running Android 4.0, as shown below:

|  |  |  |  |
| --- | --- | --- | --- |
| **App Used** | **Action Done** | **Service that grabbed the Wakelock** | **Behavior** |
| Any | UI Widget pressed (like Button or ListView item clicked) | PowerManagerService | Grabs and releases lock after ~5 seconds |
| Maps / Navigation | Launching Maps or enter Navigation directions | gps-lock | Grabs lock while using GPS |
| YouTube | Watching streamed video | PowerManagerService | Wakelock held throughout entirety of video playback |
| Music | Listen to music | PowerManagerService | Wakelock held during music playback |

**Table:** A few default Android applications demonstrate the use of wakelocks

The YouTube and Music apps are good examples showing different levels of wakelocks. The YouTube application will grab a wakelock when the user streams a video. Throughout the duration of the video, the display remains on (ignoring the system’s display settings). However, if the user presses the power button during playback, the device will suspend, causing the display to turn off and audio/video playback to cease. The Music application uses a different wakelock during audio playback. The display settings aren’t changed, so the device’s screen will turn off according to the user’s display settings. With the display off, the wakelock keeps the CPU in an active state so that audio playback can continue, even if the user presses the power button.

# Choosing a Wakelock (before the implementation)

Before diving in to how to code a wakelock, it’s important to understand the kinds of wakelocks so you can pick the right one for your application. The Android PowerManager APIs describe the different wakelock flags available that change the power state of the device:

| **Flag Value** | **CPU** | **Screen** | **Keyboard** |
| --- | --- | --- | --- |
| [**PARTIAL\_WAKE\_LOCK**](http://developer.android.com/reference/android/os/PowerManager.html#PARTIAL_WAKE_LOCK) | On | Off | Off |
| [**SCREEN\_DIM\_WAKE\_LOCK**](http://developer.android.com/reference/android/os/PowerManager.html#SCREEN_DIM_WAKE_LOCK) | On | Dim | Off |
| [**SCREEN\_BRIGHT\_WAKE\_LOCK**](http://developer.android.com/reference/android/os/PowerManager.html#SCREEN_BRIGHT_WAKE_LOCK) | On | Bright | Off |
| [**FULL\_WAKE\_LOCK**](http://developer.android.com/reference/android/os/PowerManager.html#FULL_WAKE_LOCK) | On | Bright | Bright |

**Table**: taken from the Android PowerManager API.

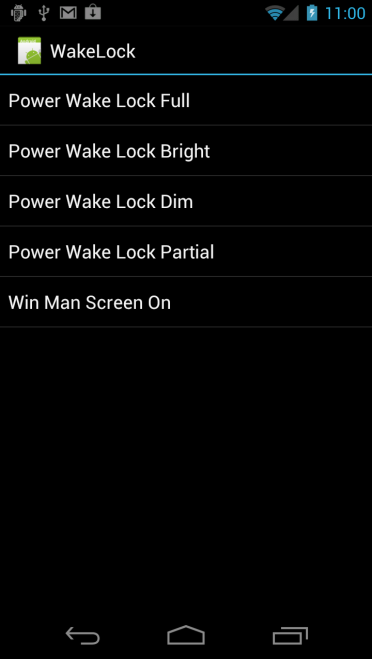
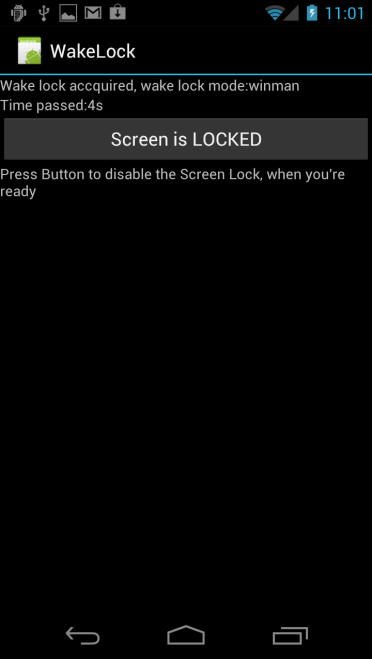
This same API strongly emphasizes that wakelocks will significantly shorten the battery life of an Android device, so they should not be used if you can avoid them. They should also be released as soon as possible.

An application that uses a wakelock must request special permission to do so. This is done with the android.permission.WAKE\_LOCK permission inside the application’s manifest file. This means that when a user installs an application using wakelocks through Google Play, they will be warned that the application contains features that will likely “Prevent phone from sleeping.”

If a developer wishes to prevent the display from dimming during a specific use case for an application, Google has added a way to do so that doesn’t require special permission. The WindowManager has a FLAG\_KEEP\_SCREEN\_ON variable that can be set when an application’s View wants to keep the screen from turning off. It’s recommended to use this approach for display control, as its effects stay isolated within the application. As soon as the user task switches to another application, the WindowManager releases the wakelock.

## Keeping the Display On (from SDPSamples)

The WakeLock application from the SDPSamples collection demonstrates (with code) how an application can keep the display on using the Window Manager, rather than writing wakelock code. Upon launching the WakeLock app, select the “Win Man Screen On” list item.

As long as the button’s state displays “Screen is LOCKED”, the screen will stay on. Once the button’s state is changed to “Screen is UNLOCKED”, the display will turn off after 5 seconds of no interaction.

In code, this was done in the screenLockUpdateState() function within WakeLockActivity.java by setting and releasing the FLAG\_KEEP\_SCREEN\_ON variable for the current window every time the button is pressed and the state changes.

**public** **void** screenLockUpdateState() {

**if** (mIsDisplayLocked)

{

...

// update display state

getWindow().addFlags(WindowManager.LayoutParams.*FLAG\_KEEP\_SCREEN\_ON*);

} **else**

{

...

// update display state

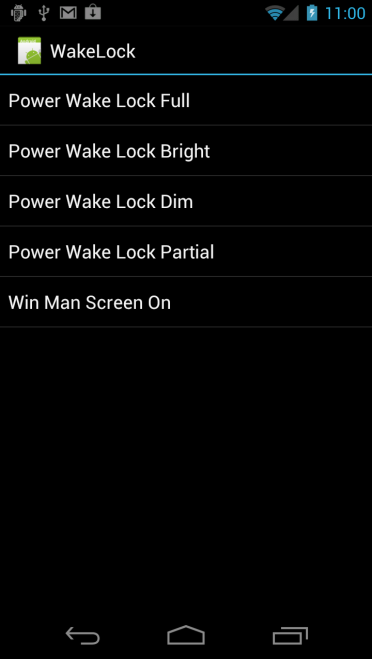
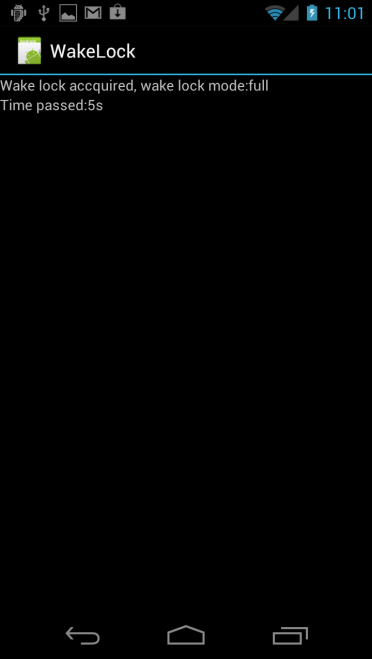
getWindow().clearFlags(WindowManager.LayoutParams.*FLAG\_KEEP\_SCREEN\_ON*);

}

}

# Coding a Wakelock (from SDPSamples)

The WakeLock application from the SDPSamples collection also has code for implementing the different types of wakelocks. Upon launching the WakeLock app, choose from the 4 different wakelock types: Power Wake Lock Full, Power Wake Lock Bright, Power Wake Lock Dim, and Power Wake Lock Partial. These 4 list items correspond to the 4 wakelock flags described in the PowerManager API. Each item will demonstrate how the device responds to an attempt to turn the screen off in 5 seconds.

Wakelocks

By monitoring the contents of the /sys/power/wake\_lock file (root access required), you can see that the PARTIAL\_WAKE\_LOCK is the only wakelock that stays in effect after the power button is pressed. The other wakelocks keep the display from turning off, with various brightness levels.

The first step to coding a wakelock is to request permission to use wakelocks within the manifest, AndroidManifest.xml:

<manifest xmlns:android=*"http://schemas.android.com/apk/res/android"*

package=*"android.intel.sdp.WakeLock"*

android:versionCode=*"1"*

android:versionName=*"1.0"* >

...

<uses-permission android:name=*"android.permission.WAKE\_LOCK"* />

...

</manifest>

After this is done, a WakeLock object can be created, which includes acquire() and release() functions to control the wakelock. A good example of this is inside the WakeLockActivity.java file:

**public** **void** onCreate(Bundle savedInstanceState) {

...

mPowerManager = (PowerManager) getSystemService(Context.*POWER\_SERVICE*);

...

mWakeLock = mPowerManager.newWakeLock(mWakeLockState,

"UMSE PowerTest");

**if** (mWakeLock != **null**) {

mWakeLock.acquire();

...

}

}

**protected** **void** onDestroy() {

**if** (mWakeLock != **null**) {

mWakeLock.release();

mWakeLock = **null**;

}

...

}

# Conclusion

A wakelock is a powerful concept in Android that allows the developer to modify the default power state of their device. The danger of using a wakelock in an application is that it will reduce the battery life of a device. Some of the clear advantages of using wakelocks are visible in various default applications by Google, like road navigation and music/video playback. For this reason it is up to the developer to decide whether or not their application model could benefit from using a wakelock.

32

# About Author

Christopher Bird joined Intel’s SSG in 2007 and contributes towards building an ecosystem behind Atom devices (phones and tablets)

# References

1. Android Reference: <http://developer.android.com/reference/android/os/PowerManager.html>

   2 LWN – “Wakelocks and the embedded problem”: <http://lwn.net/Articles/318611/>

   **Notices**

   INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.  
     
   UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.  
     
   Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.   
     
   The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request. Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.   
     
   Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: <http://www.intel.com/design/literature.htm>

   Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations, and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

   Any software source code reprinted in this document is furnished under a software license and may only be used or copied in accordance with the terms of that license.

   Intel and the Intel logo are trademarks of Intel Corporation in the US and/or other countries.

   Copyright © 2012 Intel Corporation. All rights reserved.

   \*Other names and brands may be claimed as the property of others. [↑](#endnote-ref-1)