Android

Android Platform Development Kit

- SDK
- Docs
- FAQ
- Blog
- Group
- Terms
- Report a Problem

- Documentation
  - Introduction
    - Device Requirements
  - Dev Environment Setup
    - Host System Setup
    - Getting Source Code
    - Source Code Overview
    - Build System
  - Basic Bring up
    - Building New Device
    - Bring up
    - Keymaps and Keyboard
    - Display Drivers
  - Multimedia
    - Audio Subsystem
  - Power Management
    - Power Management
  - Telephony
    - Radio Interface Layer
  - Testing
    - Instrumentation Framework
    - Instrumentation Testing

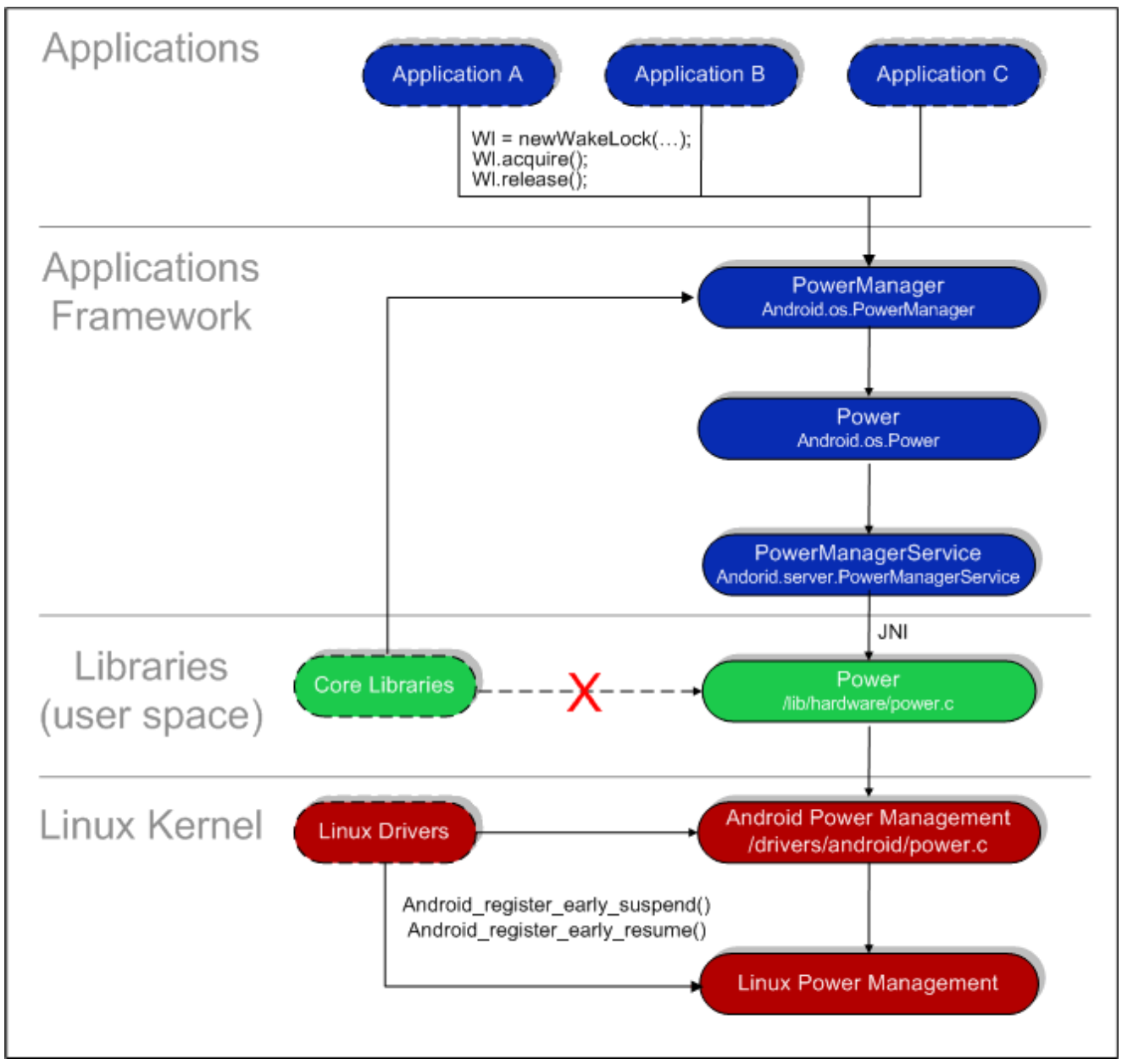
Power Management

Introduction

Android supports its own Power Management (on top of the standard Linux Power Management) designed with the premise that the CPU shouldn't consume power if no applications or services require power. For more information regarding standard Linux power management, please see Linux Power Management Support at http://kernel.org.

Android requires that applications and services request CPU resources with "wake locks" through the Android application framework and native Linux libraries. If there are no active wake locks, Android will shut down the CPU.

The image below illustrates the Android power management architecture.



Solid elements represent Android blocks and dashed elements represent partner-specific proprietary blocks.

Wake Locks

Wake locks are used by applications and services to request CPU resources.

Types of Wake Locks

Wake Lock	Description
ACQUIRE_CAUSES_WAKEUP	Normally wake locks don't actually wake the device, they just cause it to remain on once it's already on. Think of the video player app as the normal behavior. Notifications that pop up and want the device to be on are the exception; use this flag to be like them.
FULL_WAKE_LOCK	Wake lock that ensures that the screen and keyboard are on at full brightness.
ON_AFTER_RELEASE	When this wake lock is released, poke the user activity timer so the screen stays on for a little longer.
PARTIAL_WAKE_LOCK	Wake lock that ensures that the CPU is running. The screen might not be on.
SCREEN_BRIGHT_WAKE_LOCK	Wake lock that ensures that the screen is on at full brightness; the keyboard backlight will be allowed to go off.
SCREEN_DIM_WAKE_LOCK	Wake lock that ensures that the screen is on, but the keyboard backlight will be allowed to go off, and the screen backlight will be allowed to go dim.

Exploring a Wake Lock Example

All power management calls follow the same basic format:

1. Acquire handle to the PowerManager service.
2. Create a wake lock and specify the power management flags for screen, timeout, etc.
3. Acquire wake lock.
4. Perform operation (play MP3, open HTML page, etc.).
5. Release wake lock.

The snippet below illustrates this process.

```
PowerManager pm = (PowerManager)mContext.getSystemService(
                                    Context.POWER_SERVICE);
```

```
PowerManager.WakeLock wl = pm.newWakeLock(
                        PowerManager.SCREEN_DIM_WAKE_LOCK
                        | PowerManager.ON_AFTER_RELEASE,
                        TAG);
wl.acquire();
 // ...
wl.release();
```

PowerManager class

The Android Framework exposes power management to services and applications through the PowerManager class.

User space native libraries (any hardware function in //device/lib/hardware/ meant to serve as supporting libraries for Android runtime) should never call into Android Power Management directly (see the image above). Bypassing the power management policy in the Android runtime will destabilize the system.

All calls into Power Management should go through the Android runtime PowerManager APIs.

Please visit http://code.google.com/android/reference/android/os/PowerManager.html for a description of the API and examples.

Registering Drivers with the PM Driver

You can register Kernel-level drivers with the Android Power Manager driver so that they're notified immediately before power down or after power up. For example, you might set a display driver to completely power down when a request comes in to power down from the user space (see the Android MSM MDDI display driver for a sample implementation).

To register drivers with the Android PM driver, implement call-back handlers and register them with the Android PM, as illustrated in the snippet below:

```
android_register_early_suspend(android_early_suspend_t *handler)
android_register_early_resume(android_early_resume_t *handler)
```

It is critical in a drive to return immediately and not wait for anything to happen in the call back.

v0.3 - 9 June 2008