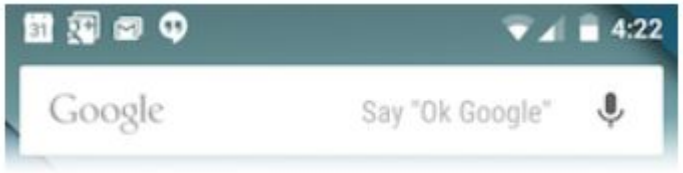
20CSI605 - Mobile Application Development

Triggering, scheduling and optimizing background tasks

Triggering, scheduling and optimizing background tasks

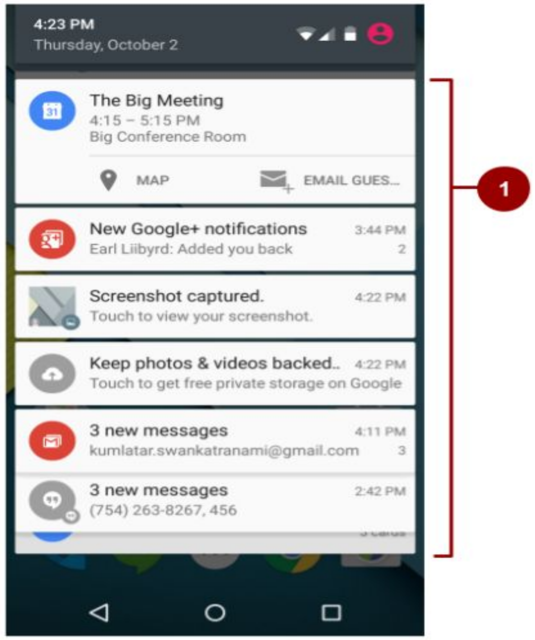
What is a notification?

- A notification is a message your app displays to the user outside your app's normal UI.
- When your app tells the system to issue a notification, the notification appears to the user as an icon in the notification area, on the left side of the status bar.



Triggering, scheduling and optimizing background tasks

- If the device is unlocked, the user opens the notification drawer to see the details of the notification.
- If the device is locked, the user views the notification on the lock screen. The notification area, lock screen, and notification drawer are system-controlled areas that the user can view at any time.


4:23 PM
Thursday, October 2

The Big Meeting
4:15 – 5:15 PM
Big Conference Room

New Google+ notifications 3:44 PM
Earl Liibyrd: Added you back 2

Screenshot captured. 4:22 PM
Touch to view your screenshot.

Keep photos & videos backed.. 4:22 PM
Touch to get free private storage on Google

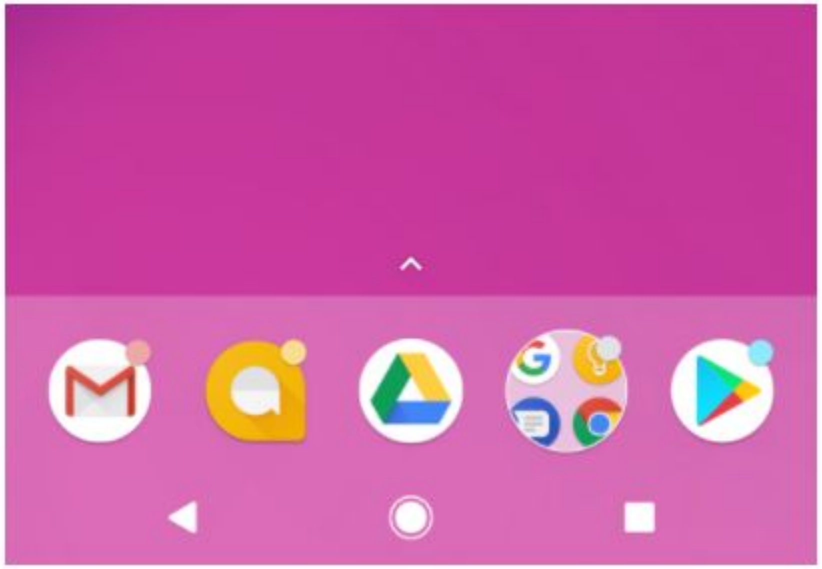
3 new messages 4:11 PM
kumlatar.swankatranami@gmail.com 3

3 new messages 2:42 PM
(754) 263-8267, 456

Triggering, scheduling and optimizing background tasks

App icon badge

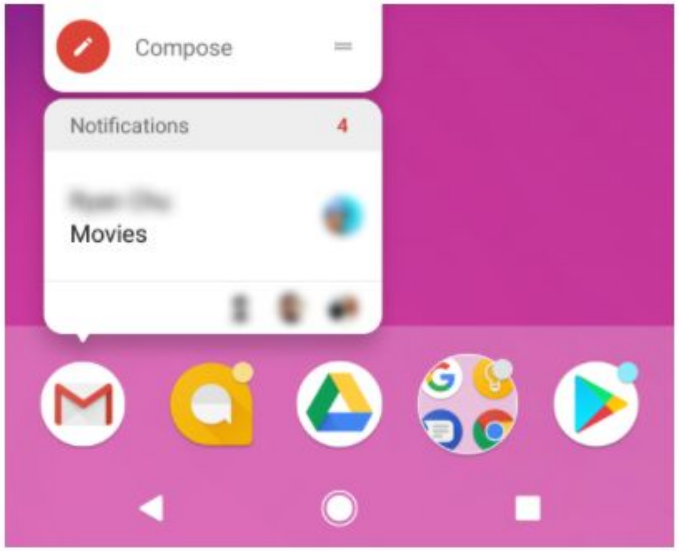
In supported launchers on devices running Android 8.0 (API level 26) and higher, an app icon changes its appearance slightly when the app has a new notification to show to the user.



Triggering, scheduling and optimizing background tasks

App icon badge

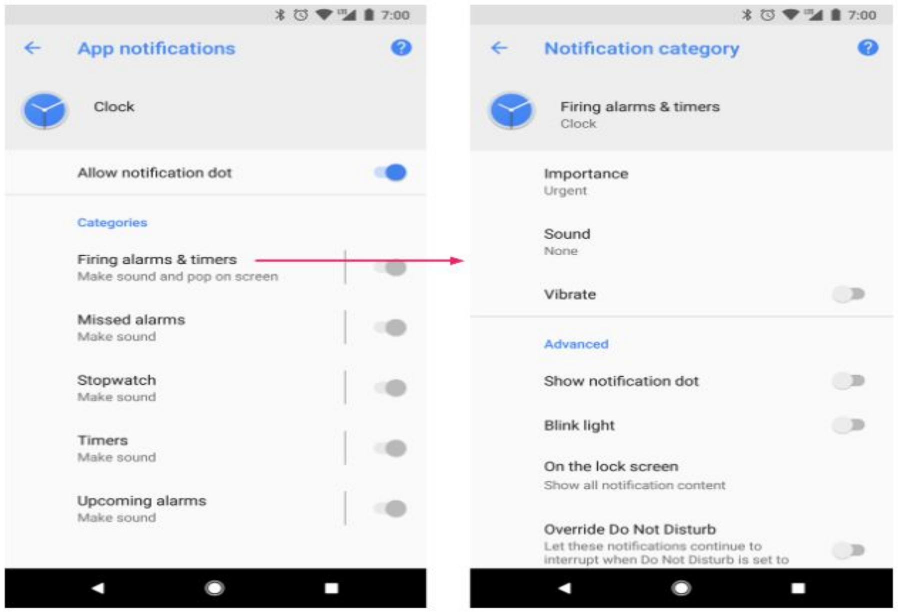
- To see the notification for an app with a notification dot, the user long-presses the app icon.
- The notification menu appears, as shown below, and the user dismisses the notification or acts on it from the menu. This is similar to the way the user interacts with a notification in the notification drawer.



Triggering, scheduling and optimizing background tasks

Notification channels

- In the Settings app on an Android-powered device, users can adjust the notifications they receive.
- Starting with Android 8.0 (API level 26), you can assign each of your app's notifications to a *notification channel*.



App notifications

Clock

Allow notification dot

Categories

Firing alarms & timers
Make sound and pop on screen

Missed alarms
Make sound

Stopwatch
Make sound

Timers
Make sound

Upcoming alarms
Make sound

Notification category

Firing alarms & timers
Clock

Importance
Urgent

Sound
None

Vibrate

Advanced

Show notification dot

Blink light

On the lock screen
Show all notification content

Override Do Not Disturb
Let these notifications continue to interrupt when Do Not Disturb is set to

Triggering, scheduling and optimizing background tasks

Creating a notification channel

To create a notification channel instance, use the Notification Channel constructor. Specify an ID that's unique within your package, a user-visible channel name, and an importance for the channel:

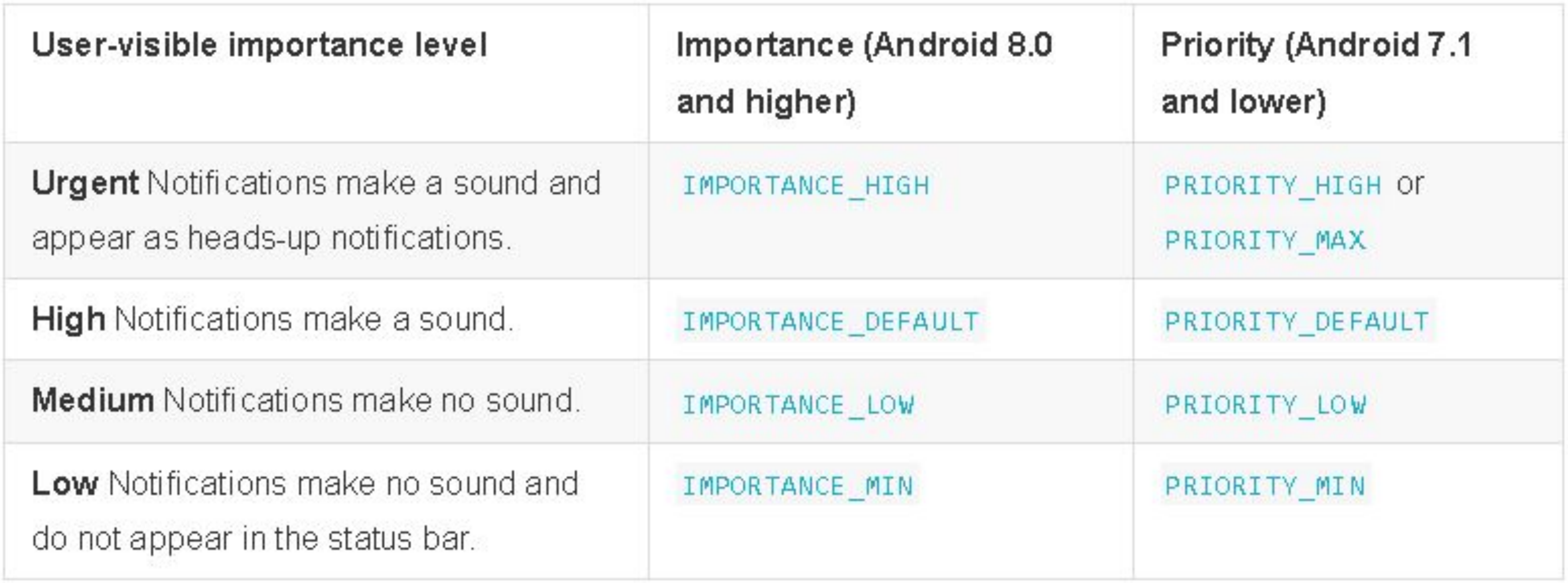
```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
    NotificationChannel notificationChannel =
            new NotificationChannel(CHANNEL_ID, "Mascot Notification",
            NotificationManager.IMPORTANCE_DEFAULT);
}
```

Set the importance level

```
mBuilder.setPriority(NotificationCompat.PRIORITY_HIGH);
```

Triggering, scheduling and optimizing background tasks

Set the importance level



User-visible importance level	Importance (Android 8.0 and higher)	Priority (Android 7.1 and lower)
Urgent Notifications make a sound and appear as heads-up notifications.	`IMPORTANCE_HIGH`	`PRIORITY_HIGH` or `PRIORITY_MAX`
High Notifications make a sound.	`IMPORTANCE_DEFAULT`	`PRIORITY_DEFAULT`
Medium Notifications make no sound.	`IMPORTANCE_LOW`	`PRIORITY_LOW`
Low Notifications make no sound and do not appear in the status bar.	`IMPORTANCE_MIN`	`PRIORITY_MIN`

Triggering, scheduling and optimizing background tasks

Configure the initial settings

✓Configure the notification channel object with initial settings such as an alert sound, a notification light color, and an optional user-visible description.

```
notificationChannel.enableLights(true);
notificationChannel.setLightColor(Color.RED);
notificationChannel.enableVibration(true);
notificationChannel.setDescription("Notification from Mascot");
```

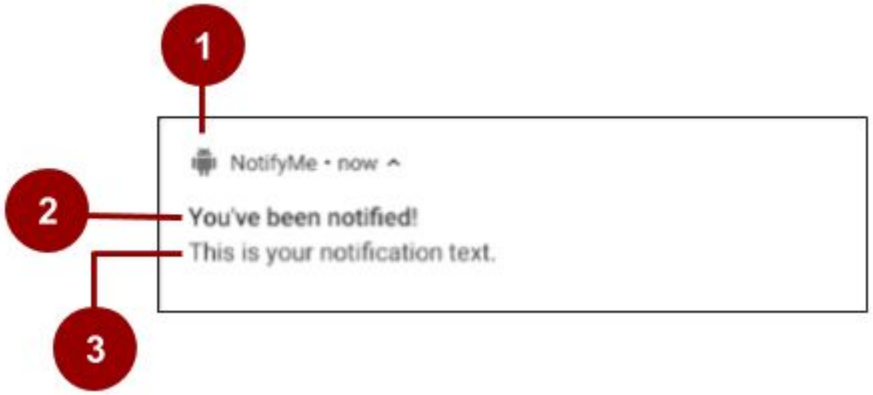
Create the notification channel

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
NotificationManager mNotificationManager =
        (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
mNotifyManager.createNotificationChannel(notificationChannel);
}
```

Triggering, scheduling and optimizing background tasks

Set notification contents

✓ You can assign components to the notification like a small icon, a title, and the notification message.



1. A small icon, set by `setSmallIcon()`. This is the only content that's required.
2. A title, set by `setContentTitle()`.
3. The body text, set by `setContentText()`. This text must be fewer than 40 characters, and it should not repeat what is in the title.

Triggering, scheduling and optimizing background tasks

Delivering notifications

Use the NotificationManager class to deliver notifications:

1. To create an instance of NotificationManager, call getSystemService(), passing in the NOTIFICATION_SERVICE constant.
2. To deliver the notification, call notify().

Pass these two values in the notify() method:

- ✓ A notification ID, which is used to update or cancel the notification.
- ✓ The NotificationCompat object that you created using the NotificationCompat.Builder object.

Triggering, scheduling and optimizing background tasks

Alarms

To create alarms, you use the AlarmManager class. Alarms in Android have the following characteristics:

- ✓ Alarms let you send an Intent at set times or intervals. You can use alarms with broadcast receivers to start services and perform other operations.
- ✓ Alarms operate outside your app. You can use them to trigger events or actions even when your app isn't running, and even if the device is asleep.
- ✓ When used correctly, alarms can help you minimize your app's resource requirements

Triggering, scheduling and optimizing background tasks

When *not* to use alarms:

- Don't use alarms for timing events such as ticks and timeouts, or for timed operations that are guaranteed to happen during the lifetime of your app.
- Don't use alarms for server sync operations.
- You might not want to use alarms for tasks that can wait until conditions are favorable, such as when the device is connected to Wi-Fi and is charging.
- Don't use alarms for tasks that have to happen even if the device is idle.

Triggering, scheduling and optimizing background tasks

Alarm types

There are two general types of alarms in Android: elapsed real-time alarms and real-time clock (RTC) alarms, and both use PendingIntent objects.

Elapsed real-time alarms

Elapsed real-time alarms use the time, in milliseconds, since the device was booted. Time zones don't affect elapsed real-time alarms, so these alarms work well for alarms based on the passage of time

The AlarmManager class provides two types of elapsed real-time alarm:

- ELAPSED_REALTIME
- ELAPSED_REALTIME_WAKEUP

Triggering, scheduling and optimizing background tasks

Real-time clock (RTC) alarms

Real-time clock (RTC) alarms are clock-based alarms that use Coordinated Universal Time (UTC).

Only choose an RTC alarm in these types of situations:

- You need your alarm to fire at a particular time of day.
- The alarm time is dependent on current locale.

The AlarmManager class provides two types of RTC alarm:

- ✓ RTC: Fires the pending intent at the specified time but doesn't wake the device.
- ✓ RTC_WAKEUP: Fires the pending intent at the specified time.

Triggering, scheduling and optimizing background tasks

Scheduling a repeating alarm

You can also use the AlarmManager to schedule repeating alarms, using one of the following methods:

- ✓ setRepeating(): On devices running Android versions lower than 4.4 (API Level 19), this method creates a repeating, exactly timed alarm.
- ✓ setInexactRepeating(): This method creates a repeating, inexact alarm that allows for batching. When you use setInexactRepeating(), Android synchronizes repeating alarms from multiple apps and fires them at the same time.

Triggering, scheduling and optimizing background tasks

Efficient data transfer

- ✓ Transferring data is an essential part of most Android apps, but it can affect battery life and increase data usage.
- ✓ Using the wireless radio to transfer data is potentially one of your app's most significant sources of battery drain.
- ✓ Users care about battery drain because they would rather use their mobile device without it connected to the charger.
- ✓ And users care about data usage, because every bit of data transferred can cost them money.

Triggering, scheduling and optimizing background tasks

Wireless radio state

- A fully active wireless radio consumes significant power. To conserve power when not in use, the radio transitions between different energy states.

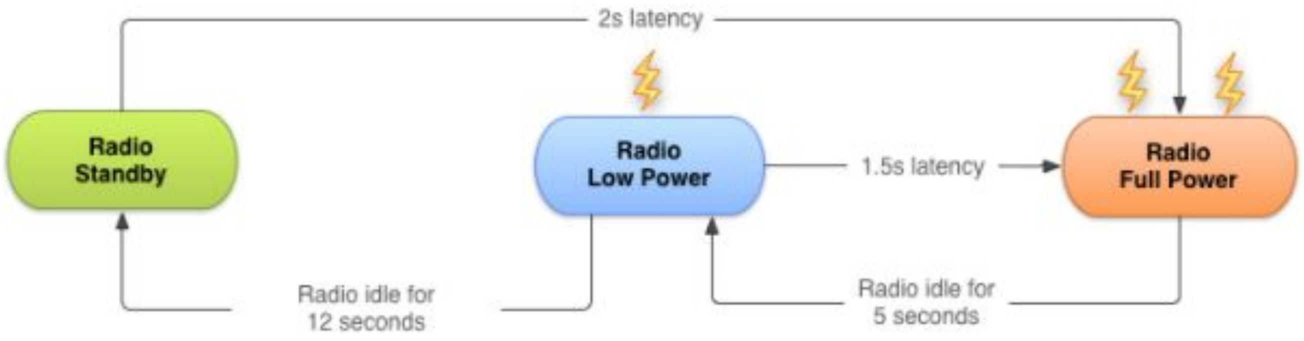
For a typical 3G network the radio has these three energy states:

- **Full power:** Used when a connection is active. Allows the device to transfer data at its highest possible rate.
- **Low power:** An intermediate state that uses about 50% less battery.
- **Standby:** The minimal energy state, during which no network connection is active or required.

Triggering, scheduling and optimizing background tasks

Wireless radio state

- Android uses a state machine to determine how to transition between states. To minimize latency, the state machine waits a short time before it transitions to lower energy states.



- The radio state machine on each device, particularly the transition delay ("tail time") and startup latency, vary based on the wireless radio technology employed, for example 2G, 3G, or LTE.

Triggering, scheduling and optimizing background tasks

Bundling network transfers

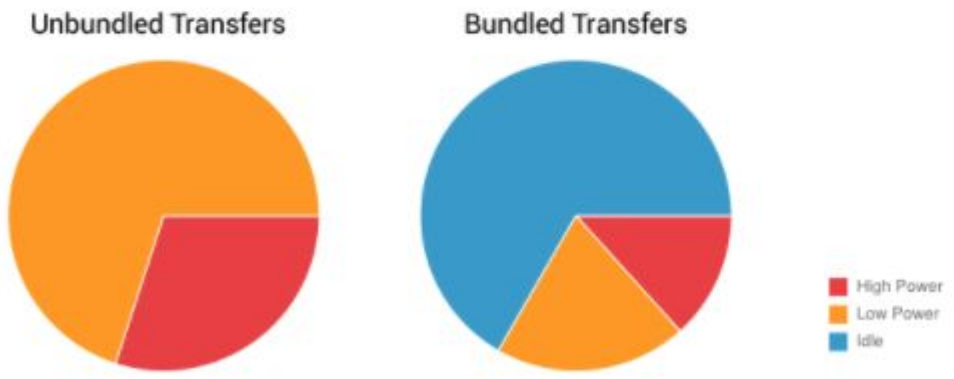
- Every time you create a new network connection, the radio transitions to the full power state.
- In the case of the 3G radio state machine described above, the radio remains at full power for the duration of your transfer.
- Then there are 5 seconds of tail time, followed by 12 seconds at the low energy state. Then the radio turns off.
- For a typical 3G device, every data transfer session causes the radio to draw power for almost 20 seconds.

Triggering, scheduling and optimizing background tasks

Bundling network transfers

What this means in practice:

- ✓ An app that transfers unbundled data for 1 second every 18 seconds keeps the wireless radio always active.
- ✓ An app that transfers bundled data for 3 seconds every minute keeps the radio in the high power state for only 8 seconds. Then the radio is in the low power state for an additional 12 seconds.



Triggering, scheduling and optimizing background tasks

Prefetching

- To *prefetch* data means that your app takes a guess at what content or data the user will want next, and fetches the data ahead of time.

For example, when the user looks at the first part of an article, a good guess is to prefetch the next part. If a user is watching a video, fetching the next minutes of the video is also a good guess.

- Prefetching allows you to download all the data you are likely to need for a given time period in a single burst, over a single connection, at full capacity.
- This reduces the number of radio activations required to download the data. As a result, you conserve battery life, improve latency for the user, lower the required bandwidth, and reduce download times.

Triggering, scheduling and optimizing background tasks

Monitor connectivity state

Devices can network using different types of hardware:

- *Wireless radios* use varying amounts of battery depending on technology, and higher bandwidth consumes more energy. Higher bandwidth means you can prefetch more aggressively, downloading more data during the same amount of time.
- *Wi-Fi radio* uses significantly less battery than wireless and offers greater bandwidth.

Triggering, scheduling and optimizing background tasks

Monitor battery state

- To minimize battery drain, monitor the state of your battery and wait for specific conditions before initiating a battery-intensive operation.
- The BatteryManager broadcasts all battery and charging details in a broadcast Intent that includes the charging status.
- To check the current battery status, examine the broadcast intent

Triggering, scheduling and optimizing background tasks

JobScheduler

- Constantly monitoring the connectivity and battery status of the device can be a challenge.
- It requires using components such as broadcast receivers, which can consume system resources even when your app isn't running.
- Because transferring data efficiently is such a common task, the Android SDK provides a class that makes efficient data transfer much easier: JobScheduler.

JobScheduler has three components:

1. JobInfo uses the builder pattern to set the conditions for the task.
2. JobService is a wrapper around the Service class where the task is actually completed.
3. JobScheduler schedules and cancels tasks.

THANK YOU