

Android Developer Fundamentals V2

Alarms and Schedulers

Lesson 8



8.1 Notifications



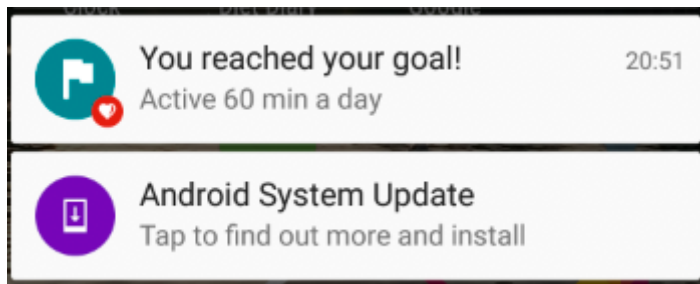
Contents

- What are notifications?
- Notification channels
- Creating a notification channel
- Creating notifications
- Tap action and action buttons
- Expanded view notifications
- Delivering notifications
- Managing Notifications

What Are Notifications?

What is a notification?

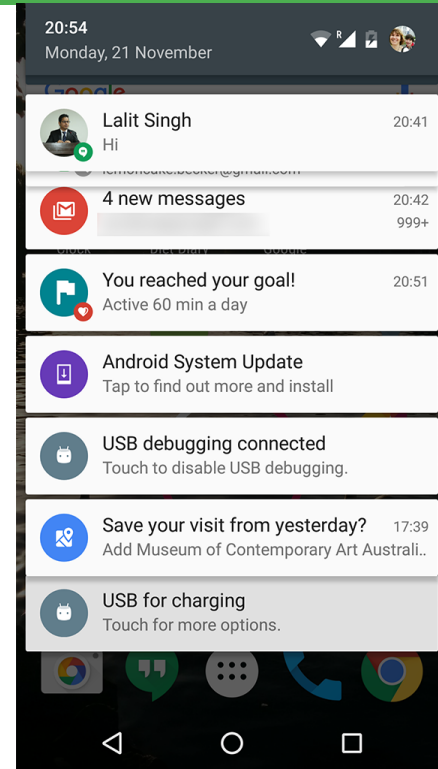
Message displayed to user outside regular app UI



- Small icon
- Title
- Detail text

How are notifications used?

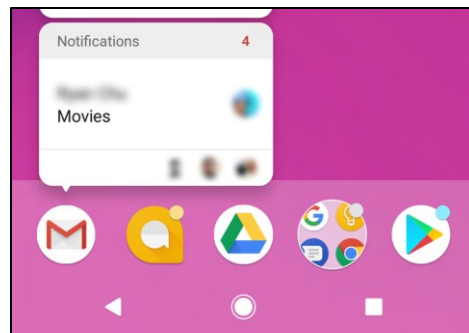
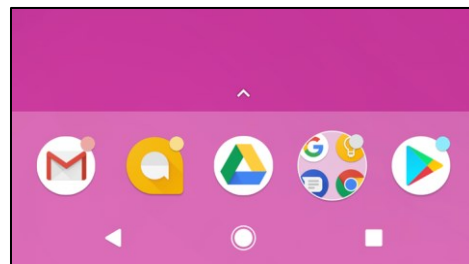
- Android issues a notification that appears as icon on the status bar.
- To see details, user opens the notification drawer.
- User can view notifications any time in the notification drawer.



App icon badge

Available only on the devices running Android 8.0 (API level 26) and higher.

- New notifications are displayed as a colored "badge" (also known as a "notification dot") on the app icon.
- Users can long-press on an app icon to see the notifications for that app. Similar to the notification drawer.



Notification Channels

Notification channels

- Used to create a user-customizable channel for each type of notification to be displayed.
- More than one notification can be grouped in to a channel.
- Set notification behavior like sound, light, vibrate and so on, applied to all the notifications in that channel.



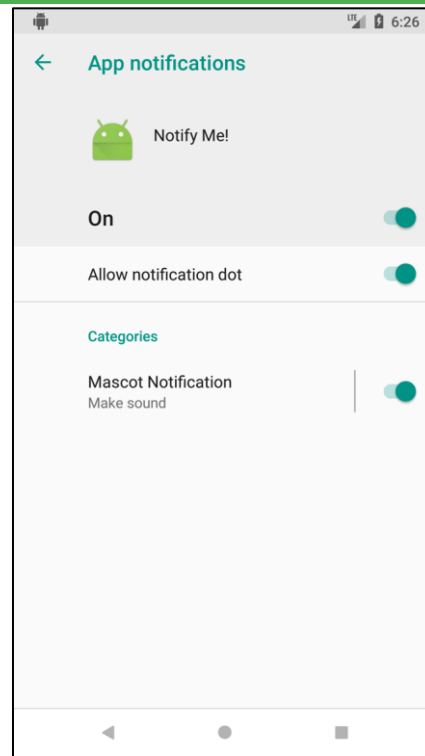
Notification channels are mandatory

- [Notification channels](#) are introduced in Android 8.0 (API level 26)
- All notifications must be assigned to a channel starting from Android 8.0 (API level 26), else your notifications will not be displayed.
- For the apps targeting lower than Android 8.0 (API level 26), no need to implement notification channels.



Notification channels in Settings

- Notification channels appear as **Categories** under **App notifications** in the device Settings.



Creating a Notification channel

Create a Notification channel

- Notification channel instance is created using [NotificationChannel](#) constructor.
- You must specify:
 - An ID that's unique within your package.
 - User visible name of the channel.
 - The importance level for the channel.

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



Importance level

- Available in Android 8.0 (API level 26) and higher.
- Sets the intrusion level, like the sound and visibility for all notifications posted in the channel.
- Range from [IMPORTANCE_NONE \(0\)](#) to [IMPORTANCE_HIGH \(4\)](#).
- To support earlier versions of Android (Lower than API level 26), set the priority.



Notification priority

- Determines how the system displays the notification with respect to other notifications, in Android version Lower than API level 26.
- Set using the `setPriority()` method for each notification.
- Range from `PRIORITY_MIN` to `PRIORITY_MAX`.

```
setPriority(NotificationCompat.PRIORITY_HIGH)
```



Importance level and priority constants

User-visible importance level	Importance (Android 8.0 and higher)	Priority (Android 7.1 and lower)
Urgent Makes a sound and appears as a heads-up notification	<u>IMPORTANCE_HIGH</u>	<u>PRIORITY_HIGH</u> or <u>PRIORITY_MAX</u>
High Makes a sound	<u>IMPORTANCE_DEFAULT</u>	<u>PRIORITY_DEFAULT</u>
Medium No sound	<u>IMPORTANCE_LOW</u>	<u>PRIORITY_LOW</u>
Low No sound and doesn't appear in the status bar	<u>IMPORTANCE_MIN</u>	<u>PRIORITY_MIN</u>



Creating Notifications

Creating Notification

- Notification is created using [NotificationCompat.Builder](#) class.
- Pass the application context and notification channel ID to the constructor.
- The [NotificationCompat.Builder](#) constructor takes the notification channel ID, this is only used by Android 8.0 (API level 26) and higher, but this parameter is ignored by the older versions.

```
NotificationCompat.Builder mBuilder = new  
    NotificationCompat.Builder(this, CHANNEL_ID);
```

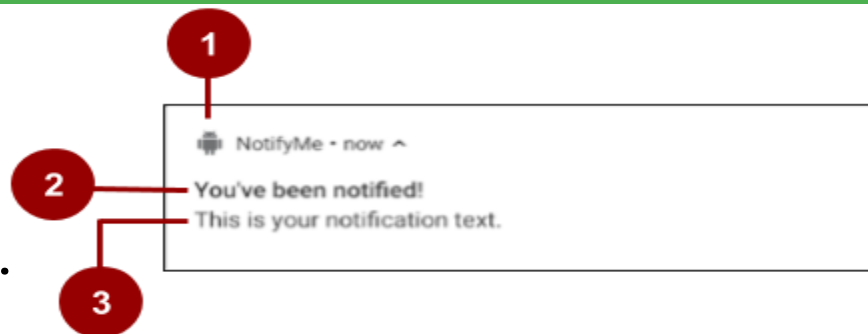


Setting notification contents

1. A small icon, set by `setSmallIcon()`.

This is the only content that's required.

1. A title, set by `setContentTitle()`.
2. The body text, set by `setContentText()`. This is the notification message.



Setting notification contents

```
NotificationCompat.Builder mBuilder =  
    new NotificationCompat.Builder(this, CHANNEL_ID)  
        .setSmallIcon(R.drawable.android_icon)  
        .setContentTitle("You've been notified!")  
        .setContentText("This is your notification text.");
```



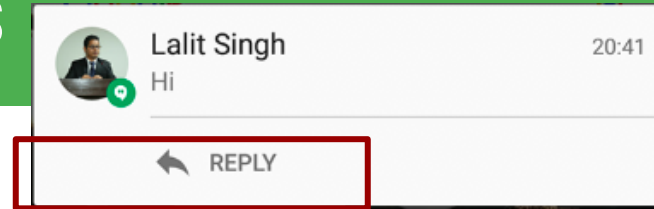
Tap action and Action buttons

Add notification tap action

- Every notification must respond when it is tapped, usually launching an Activity in your app.
- Set an content intent using `setContentIntent\(\)` method.
- Pass the `Intent` wrapped in a `PendingIntent` object.



Notification action buttons



- Action buttons can perform a variety of actions on behalf of your app, such as starting a background task, placing a phone call and so on.
- Starting from Android 7.0 (API level 24) reply to messages directly from notifications.
- To add an action button, pass a `PendingIntent` to the [`addAction\(\)`](#) method.

Pending intents

- A [PendingIntent](#) is a description of an intent and target action to perform with it.
- Give a `PendingIntent` to another application to grant it the right to perform the operation you have specified as if the other app was yourself.



Methods to create a PendingIntent

To instantiate a PendingIntent, use one of the following methods:

- [PendingIntent.getActivity\(\)](#)
- [PendingIntent.getBroadcast\(\)](#)
- [PendingIntent.getService\(\)](#)



PendingIntent method arguments

1. Application context
2. Request code—constant integer id for the pending intent
3. Intent to be delivered
4. [PendingIntent flag](#) determines how the system handles multiple pending intents from same app



Step 1: Create intent

```
Intent notificationIntent =  
    new Intent(this, MainActivity.class);
```



Step 2: Create PendingIntent

```
PendingIntent notificationPendingIntent =  
    PendingIntent.getActivity(  
        this,  
        NOTIFICATION_ID,  
        notificationIntent,  
        PendingIntent.FLAG_UPDATE_CURRENT);
```



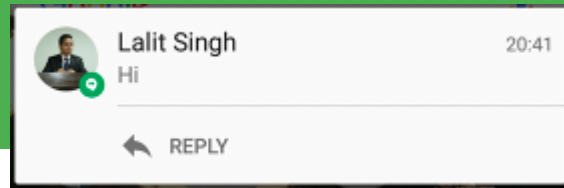
Step 3: Add to notification builder

To set tap action to the notification:

```
.setContentIntent(notificationPendingIntent);
```



Add action buttons



- Use `NotificationCompat.Builder.addAction()`
 - pass in icon, caption, `PendingIntent`

```
.addAction(R.drawable.ic_color_lens_black_24dp,  
           "R.string.label",  
           notificationPendingIntent);
```

Expanded view notifications

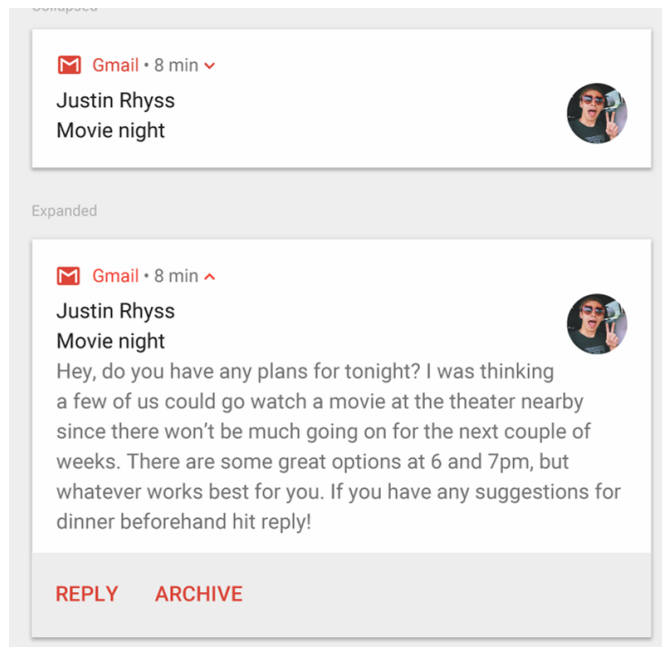
Expandable notifications

- Notifications in the notification drawer appear in two main layouts, normal view (which is the default) and expanded view.
- Expanded view notifications were introduced in Android 4.1.
- Use them sparingly -- they take up more space and attention.



Big text

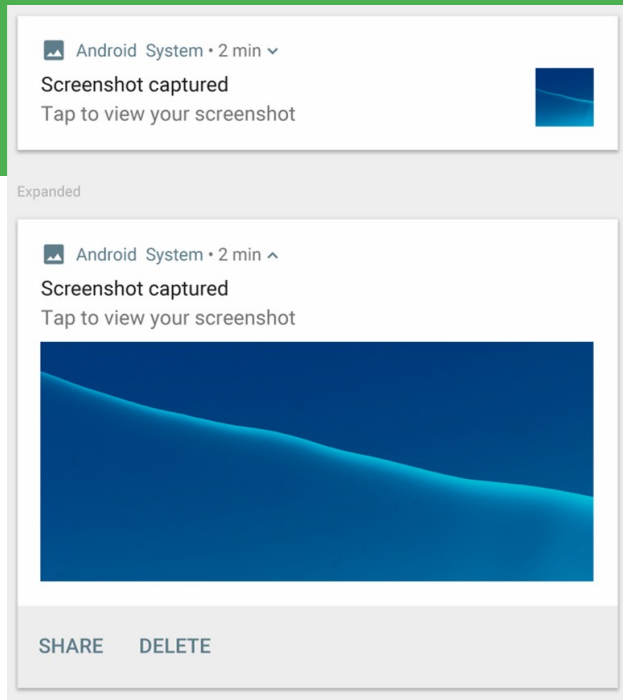
- For large-format notifications that include a lot of text.
- Fits more text than a standard view.
- Use the helper class:
[NotificationCompat.BigTextStyle](#)



Big image

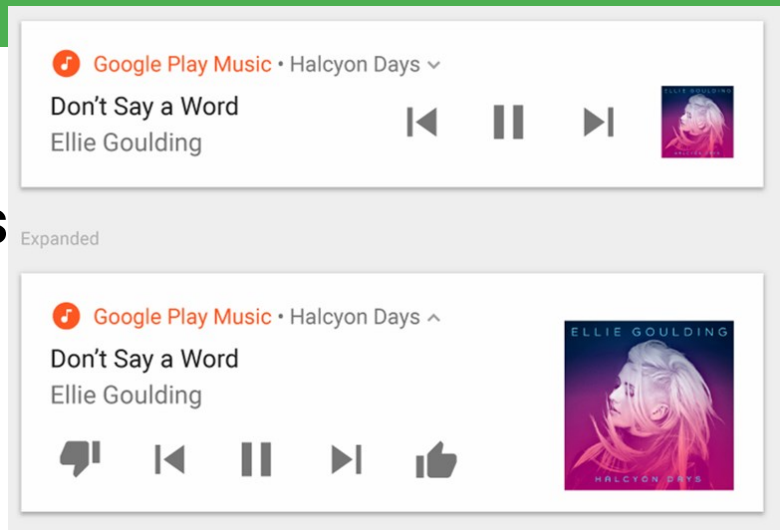
- For large-format notifications that include a large image attachment.
- Use the helper class:

[NotificationCompat.BigPictureStyle](#)



Media

- For media playback notifications
- Actions for controlling media such as music
- Image for album cover
- Use the helper class:
- [NotificationCompat.MediaStyle](#)



Setting styles

To create expandable notification that appear, use one of the helper classes to set the style using the [setStyle\(\)](#) method.

```
mNotifyBuilder
```

```
.setStyle(new NotificationCompat.BigPictureStyle\(\)  
    .bigPicture(myBitmapImage)  
    .setBigContentTitle("Notification!"));
```



Delivering Notifications

Delivering notifications

- Use the [NotificationManager](#) class to deliver notifications.
 - Create an instance of `NotificationManager`
 - Call `notify()` to deliver the notification.



Instantiate NotificationManager

Call `getSystemService()`, passing in the [NOTIFICATION_SERVICE](#) constant.

```
mNotifyManager = (NotificationManager)
    getSystemService(NOTIFICATION_SERVICE);
```



Send notification

- **Call `notify()` to deliver the notification, passing in these two values:**
 - **A notification ID, which is used to update or cancel the notification.**
 - **The `NotificationCompat` object that you created using the `NotificationCompat.Builder` object.**

```
mNotifyManager.notify(NOTIFICATION_ID, myNotification);
```



Managing Notifications

Updating notifications

1. Update a notification by changing and or adding some of its content.
2. Issue notification with updated parameters using builder.
3. Call `notify()` passing in the same notification ID.
 - If previous notification is still visible, system updates.
 - If previous notification has been dismissed, new notification is delivered.



Canceling notifications

Notifications remain visible until:

- User dismisses it by swiping or by using "Clear All".
- Calling `setAutoCancel()` when creating the notification, removes it from the status bar when the user clicks on it.
- App calls `cancel()` or `cancelAll()` on `NotificationManager`.

```
mNotifyManager.cancel(NOTIFICATION_ID);
```



Design guidelines

If your app sends too many notifications, users will disable notifications or uninstall the app.

- **Relevant:** Whether this information is essential for the user.
- **Timely:** Notifications need to appear when they are useful.
- **Short:** Use as few words as possible.
- **Give users the power to choose -- Use appropriate notification channels to categorise your notifications**

What's Next?

- Concept Chapter: [8.1 Notifications](#)
- Practical: [8.1 Notifications](#)

The End