Notifications

A notification is a message you can display to the user outside of your application's normal UI. When you tell the system to issue a notification, it first appears as an icon in the notification area. To see the details of the notification, the user opens the notification drawer. Both the notification area and the notification drawer are system-controlled areas that the user can view at any time.

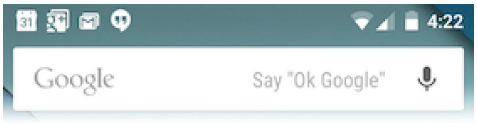

Figure 1. Notifications in the notification area.

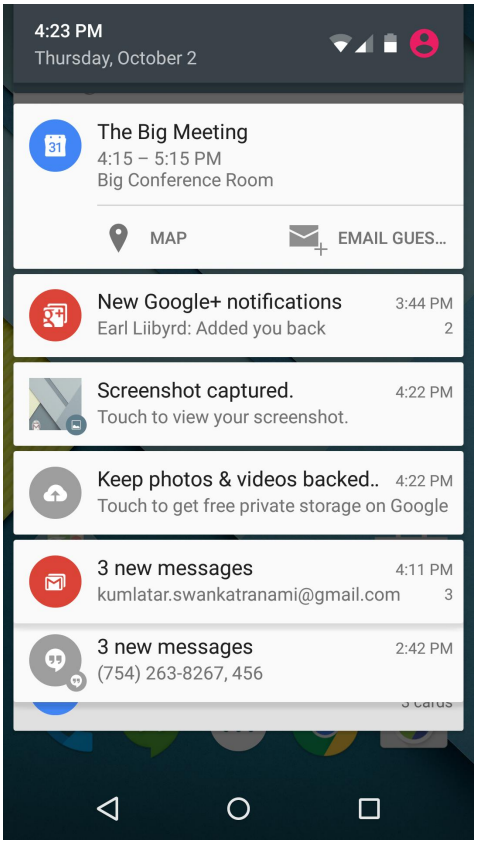

Figure 2. Notifications in the notification drawer.

> Note: Except where noted, this guide refers to the `NotificationCompat.Builder` (/reference/android/support/v4/app/NotificationCompat.Builder.html) class in the version 4 Support Library (/tools/support-library/index.html). The class `Notification.Builder` (/reference/android/app/Notification.Builder.html) was added in Android 3.0 (API level 11).

Design Considerations

Notifications, as an important part of the Android user interface, have their own design guidelines. The material design changes introduced in Android 5.0 (API level 21) are of particular importance, and you should review the Material Design

IN THIS DOCUMENT

KEY CLASSES

(/training/material/index.html) training for more information. To learn how to design notifications and their interactions, read the Notifications (/design/patterns/notifications.html) design guide.

VIDEOS

Notifications in 4.1

SEE ALSO

Android Design: Notifications

Creating a Notification

You specify the UI information and actions for a notification in a NotificationCompat.Builder (/reference/android/support/v4/app/NotificationCompat.Builder.html) object. To create the notification itself, you call NotificationCompat.Builder.build() (/reference/android/support/v4/app/NotificationCompat.Builder.html#build()), which returns a Notification (/reference/android/app/Notification.html) object containing your specifications. To issue the notification, you pass the Notification (/reference/android/app/Notification.html) object to the system by calling NotificationManager.notify() (/reference/java/lang/Object.html#notify()).

Required notification contents

A Notification (/reference/android/app/Notification.html) object *must* contain the following:

- A small icon, set by setSmallIcon()
- A title, set by setContentTitle()
- Detail text, set by setContentText()

Optional notification contents and settings

All other notification settings and contents are optional. To learn more about them, see the reference documentation for NotificationCompat.Builder (/reference/android/support/v4/app/NotificationCompat.Builder.html).

Notification actions

Although they're optional, you should add at least one action to your notification. An action allows users to go directly from the notification to an Activity (/reference/android/app/Activity.html) in your application, where they can look at one or more events or do further work.

A notification can provide multiple actions. You should always define the action that's triggered when the user clicks the notification; usually this action opens an Activity (/reference/android/app/Activity.html) in your application. You can also add buttons to the notification that perform additional actions such as snoozing an alarm or responding immediately to a text message; this feature is available as of Android 4.1. If you use additional action buttons, you must also make their functionality available in an Activity (/reference/android/app/Activity.html) in your app; see the section Handling compatibility (#Compatibility) for more details.

Inside a Notification (/reference/android/app/Notification.html), the action itself is defined by a PendingIntent (/reference/android/app/PendingIntent.html) containing an Intent (/reference/android/content/Intent.html) that starts an Activity (/reference/android/app/Activity.html) in your application. To associate the PendingIntent (/reference/android/app/PendingIntent.html) with a gesture, call the appropriate method of NotificationCompat.Builder (/reference/android/support/v4/app/NotificationCompat.Builder.html). For example, if you want to start Activity (/reference/android/app/Activity.html) when the user clicks the notification text in the notification drawer, you add the PendingIntent (/reference/android/app/PendingIntent.html) by calling setContentIntent() (/reference/android/support/v4/app/NotificationCompat.Builder.html#setContentIntent(android.app.PendingIntent)).

Starting an Activity (/reference/android/app/Activity.html) when the user clicks the notification is the most common action scenario. You can also start an Activity (/reference/android/app/Activity.html) when the user dismisses a notification. In Android 4.1 and later, you can start an Activity (/reference/android/app/Activity.html) from an action button. To learn more, read the reference guide for NotificationCompat.Builder (/reference/android/support/v4/app/NotificationCompat.Builder.html).

Notification priority

If you wish, you can set the priority of a notification. The priority acts as a hint to the device UI about how the notification should be displayed. To set a notification's priority, call NotificationCompat.Builder.setPriority() (/reference/android/support/v4/app/NotificationCompat.Builder.html#setPriority(int)) and pass in one of the NotificationCompat (/reference/android/support/v4/app/NotificationCompat.html) priority constants. There are five priority levels, ranging from PRIORITY_MIN (/reference/android/support/v4/app/NotificationCompat.html#PRIORITY_MIN) (-2) to PRIORITY_MAX (/reference/android/support/v4/app/NotificationCompat.html#PRIORITY_MAX) (2); if not set, the priority defaults to PRIORITY_DEFAULT (/reference/android/support/v4/app/NotificationCompat.html#PRIORITY_DEFAULT) (0).

For information about setting an appropriate priority level, see "Correctly set and manage notification priority" in the Notifications (/design/patterns/notifications.html) Design guide.

Creating a simple notification

The following snippet illustrates a simple notification that specifies an activity to open when the user clicks the notification. Notice that the code creates a TaskStackBuilder (/reference/android/support/v4/app/TaskStackBuilder.html) object and uses it to create the PendingIntent (/reference/android/app/PendingIntent.html) for the action. This pattern is explained in more detail in the section Preserving Navigation when Starting an Activity (#NotificationResponse):

```
NotificationCompat.Builder mBuilder =
        new NotificationCompat.Builder(this)
        .setSmallIcon(R.drawable.notification_icon)
        .setContentTitle("My notification")
        .setContentText("Hello World!");
// Creates an explicit intent for an Activity in your app
Intent resultIntent = new Intent(this, ResultActivity.class);

// The stack builder object will contain an artificial back stack for the
// started Activity.
// This ensures that navigating backward from the Activity leads out of
// your application to the Home screen.
TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
// Adds the back stack for the Intent (but not the Intent itself)
stackBuilder.addParentStack(ResultActivity.class);
// Adds the Intent that starts the Activity to the top of the stack
stackBuilder.addNextIntent(resultIntent);
PendingIntent resultPendingIntent =
        stackBuilder.getPendingIntent(
            0,
            PendingIntent.FLAG_UPDATE_CURRENT
        );
mBuilder.setContentIntent(resultPendingIntent);
NotificationManager mNotificationManager =
    (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
// mId allows you to update the notification later on.
mNotificationManager.notify(mId, mBuilder.build());
```

That's it. Your user has now been notified.

Applying an expanded layout to a notification

To have a notification appear in an expanded view, first create a NotificationCompat.Builder (/reference/android/support/v4/app/NotificationCompat.Builder.html) object with the normal view options you want. Next, call Builder.setStyle() (/reference/android/support/v4/app/NotificationCompat.Builder.html#setStyle(android.support.v4.app.NotificationCompat.Style)) with an expanded layout object as its argument.

Remember that expanded notifications are not available on platforms prior to Android 4.1. To learn how to

handle notifications for Android 4.1 and for earlier platforms, read the section Handling compatibility (#Compatibility).

For example, the following code snippet demonstrates how to alter the notification created in the previous snippet to use the expanded layout:

```
NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(this)
    .setSmallIcon(R.drawable.notification_icon)
    .setContentTitle("Event tracker")
    .setContentText("Events received")
NotificationCompat.InboxStyle inboxStyle =
        new NotificationCompat.InboxStyle();
String[] events = new String[6];
// Sets a title for the Inbox in expanded layout
inboxStyle.setBigContentTitle("Event tracker details:");
...
// Moves events into the expanded layout
for (int i=0; i < events.length; i++) {

    inboxStyle.addLine(events[i]);
}
// Moves the expanded layout object into the notification object.
mBuilder.setStyle(inBoxStyle);
...
// Issue the notification here.
```

Handling compatibility

Not all notification features are available for a particular version, even though the methods to set them are in the support library class NotificationCompat.Builder (/reference/android/support/v4/app/NotificationCompat.Builder.html). For example, action buttons, which depend on expanded notifications, only appear on Android 4.1 and higher, because expanded notifications themselves are only available on Android 4.1 and higher.

To ensure the best compatibility, create notifications with NotificationCompat (/reference/android/support/v4/app/NotificationCompat.html) and its subclasses, particularly NotificationCompat.Builder (/reference/android/support/v4/app/NotificationCompat.Builder.html). In addition, follow this process when you implement a notification:

1. Provide all of the notification's functionality to all users, regardless of the version they're using. To do this, verify that all of the functionality is available from an Activity in your app. You may want to add a new Activity to do this.

   For example, if you want to use addAction() (/reference/android/support/v4/app/NotificationCompat.Builder.html#addAction(android.support.v4.app.NotificationCompat.Action)) to provide a control that stops and starts media playback, first implement this control in an Activity (/reference/android/app/Activity.html) in your app.

2. Ensure that all users can get to the functionality in the Activity, by having it start when users click the notification. To do this, create a PendingIntent for the Activity. Call setContentIntent() to add the PendingIntent to the notification.
3. Now add the expanded notification features you want to use to the notification. Remember that any functionality you add also has to be available in the Activity that starts when users click the notification.

Managing Notifications

When you need to issue a notification multiple times for the same type of event, you should avoid making a completely new notification. Instead, you should consider updating a previous notification, either by changing some of its values or by adding to it, or both.

For example, Gmail notifies the user that new emails have arrived by increasing its count of unread messages and by adding a summary of each email to the notification. This is called "stacking" the notification; it's described in more detail in the Notifications (/design/patterns/notifications.html) Design guide.

> Note: This Gmail feature requires the "inbox" expanded layout, which is part of the expanded notification feature available starting in Android 4.1.

The following section describes how to update notifications and also how to remove them.

Updating notifications

To set up a notification so it can be updated, issue it with a notification ID by calling NotificationManager.notify() (/reference/android/app/NotificationManager.html#notify(int, android.app.Notification)). To update this notification once you've issued it, update or create a NotificationCompat.Builder (/reference/android/support/v4/app/NotificationCompat.Builder.html) object, build a Notification (/reference/android/app/Notification.html) object from it, and issue the Notification (/reference/android/app/Notification.html) with the same ID you used previously. If the previous notification is still visible, the system updates it from the contents of the Notification (/reference/android/app/Notification.html) object. If the previous notification has been dismissed, a new notification is created instead.

The following snippet demonstrates a notification that is updated to reflect the number of events that have occurred. It stacks the notification, showing a summary:

```
mNotificationManager =
        (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
// Sets an ID for the notification, so it can be updated
int notifyID = 1;
mNotifyBuilder = new NotificationCompat.Builder(this)
    .setContentTitle("New Message")
    .setContentText("You've received new messages.")
    .setSmallIcon(R.drawable.ic_notify_status)
numMessages = 0;
// Start of a loop that processes data and then notifies the user
...
    mNotifyBuilder.setContentText(currentText)
        .setNumber(++numMessages);
    // Because the ID remains unchanged, the existing notification is
    // updated.
    mNotificationManager.notify(
            notifyID,
            mNotifyBuilder.build());
...
```

Removing notifications

Notifications remain visible until one of the following happens:

- The user dismisses the notification either individually or by using "Clear All" (if the notification can be cleared).
- The user clicks the notification, and you called setAutoCancel() when you created the notification.
- You call cancel() for a specific notification ID. This method also deletes ongoing notifications.
- You call cancelAll(), which removes all of the notifications you previously issued.

Preserving Navigation when Starting an Activity

When you start an Activity (/reference/android/app/Activity.html) from a notification, you must preserve the user's expected navigation experience. Clicking *Back* should take the user back through the application's normal work flow to the Home screen, and clicking *Recents* should show the Activity (/reference/android/app/Activity.html) as a separate task. To preserve the navigation experience, you should

start the Activity (/reference/android/app/Activity.html) in a fresh task. How you set up the PendingIntent (/reference/android/app/PendingIntent.html) to give you a fresh task depends on the nature of the Activity (/reference/android/app/Activity.html) you're starting. There are two general situations:

Regular activity
: You're starting an Activity that's part of the application's normal workflow. In this situation, set up the PendingIntent to start a fresh task, and provide the PendingIntent with a back stack that reproduces the application's normal *Back* behavior.

  Notifications from the Gmail app demonstrate this. When you click a notification for a single email message, you see the message itself. Touching **Back** takes you backwards through Gmail to the Home screen, just as if you had entered Gmail from the Home screen rather than entering it from a notification.

  This happens regardless of the application you were in when you touched the notification. For example, if you're in Gmail composing a message, and you click a notification for a single email, you go immediately to that email. Touching *Back* takes you to the inbox and then the Home screen, rather than taking you to the message you were composing.

Special activity
: The user only sees this Activity if it's started from a notification. In a sense, the Activity extends the notification by providing information that would be hard to display in the notification itself. For this situation, set up the PendingIntent to start in a fresh task. There's no need to create a back stack, though, because the started Activity isn't part of the application's activity flow. Clicking *Back* will still take the user to the Home screen.

Setting up a regular activity PendingIntent

To set up a PendingIntent (/reference/android/app/PendingIntent.html) that starts a direct entry Activity (/reference/android/app/Activity.html), follow these steps:

1. Define your application's Activity hierarchy in the manifest.
   a. Add support for Android 4.0.3 and earlier. To do this, specify the parent of the Activity you're starting by adding a <meta-data> element as the child of the <activity>.

      For this element, set android:name (/guide/topics/manifest/meta-data-element.html#nm)="android.support.PARENT_ACTIVITY". Set android:value (/guide/topics/manifest/meta-data-element.html#val)="<parent_activity_name>" where <parent_activity_name> is the value of android:name (/guide/topics/manifest/meta-data-element.html#nm) for the parent <activity> (/guide/topics/manifest/activity-element.html) element. See the following XML for an example.

   b. Also add support for Android 4.1 and later. To do this, add the android:parentActivityName attribute to the <activity> element of the Activity you're starting.

   The final XML should look like this:

```
<activity
    android:name=".MainActivity"
    android:label="@string/app_name" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
<activity
    android:name=".ResultActivity"
    android:parentActivityName=".MainActivity">
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value=".MainActivity"/>
</activity>
```

2. Create a back stack based on the Intent that starts the Activity:
    a. Create the Intent to start the Activity.
    b. Create a stack builder by calling TaskStackBuilder.create().
    c. Add the back stack to the stack builder by calling addParentStack(). For each Activity in the hierarchy you've defined in the manifest, the back stack contains an Intent object that starts the Activity. This method also adds flags that start the stack in a fresh task.

    > Note: Although the argument to addParentStack() (/reference/android/support/v4/app/TaskStackBuilder.html#addParentStack(android.app.Activity)) is a reference to the started Activity (/reference/android/app/Activity.html), the method call doesn't add the Intent (/reference/android/content/Intent.html) that starts the Activity (/reference/android/app/Activity.html). Instead, that's taken care of in the next step.

    d. Add the Intent that starts the Activity from the notification, by calling addNextIntent(). Pass the Intent you created in the first step as the argument to addNextIntent().
    e. If you need to, add arguments to Intent objects on the stack by calling TaskStackBuilder.editIntentAt(). This is sometimes necessary to ensure that the target Activity displays meaningful data when the user navigates to it using *Back*.
    f. Get a PendingIntent for this back stack by calling getPendingIntent(). You can then use this PendingIntent as the argument to setContentIntent().

The following code snippet demonstrates the process:

```
...
Intent resultIntent = new Intent(this, ResultActivity.class);
TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
// Adds the back stack
stackBuilder.addParentStack(ResultActivity.class);
// Adds the Intent to the top of the stack
stackBuilder.addNextIntent(resultIntent);
// Gets a PendingIntent containing the entire back stack
PendingIntent resultPendingIntent =
        stackBuilder.getPendingIntent(0, PendingIntent.FLAG_UPDATE_CURRENT);
...
NotificationCompat.Builder builder = new NotificationCompat.Builder(this);
builder.setContentIntent(resultPendingIntent);
NotificationManager mNotificationManager =
    (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
mNotificationManager.notify(id, builder.build());
```

Setting up a special activity PendingIntent

The following section describes how to set up a special activity PendingIntent (/reference/android/app/PendingIntent.html).

A special Activity (/reference/android/app/Activity.html) doesn't need a back stack, so you don't have to define its Activity (/reference/android/app/Activity.html) hierarchy in the manifest, and you don't have to call addParentStack() (/reference/android/support/v4/app/TaskStackBuilder.html#addParentStack(android.app.Activity)) to build a back stack. Instead, use the manifest to set up the Activity (/reference/android/app/Activity.html) task options, and create the PendingIntent (/reference/android/app/PendingIntent.html) by calling getActivity() (/reference/android/app/PendingIntent.html#getActivity(android.content.Context, int, android.content.Intent, int)):

1. In your manifest, add the following attributes to the <activity> element for the Activity

    android:name="activityclass"
        The activity's fully-qualified class name.
    android:taskAffinity=""
        Combined with the FLAG_ACTIVITY_NEW_TASK flag that you set in code, this ensures that this Activity doesn't go into the application's default task. Any existing tasks that have the application's default affinity are not affected.
    android:excludeFromRecents="true"

Excludes the new task from *Recents*, so that the user can't accidentally navigate back to it.

This snippet shows the element:

```
<activity
    android:name=".ResultActivity"
...
    android:launchMode="singleTask"
    android:taskAffinity=""
    android:excludeFromRecents="true">
</activity>
...
```

2. Build and issue the notification:
   a. Create an `Intent` that starts the `Activity`.
   b. Set the `Activity` to start in a new, empty task by calling `setFlags()` with the flags `FLAG_ACTIVITY_NEW_TASK` and `FLAG_ACTIVITY_CLEAR_TASK`.
   c. Set any other options you need for the `Intent`.
   d. Create a `PendingIntent` from the `Intent` by calling `getActivity()`. You can then use this `PendingIntent` as the argument to `setContentIntent()`.

   The following code snippet demonstrates the process:

```
// Instantiate a Builder object.
NotificationCompat.Builder builder = new NotificationCompat.Builder(this);
// Creates an Intent for the Activity
Intent notifyIntent =
        new Intent(this, ResultActivity.class);
// Sets the Activity to start in a new, empty task
notifyIntent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK
                        | Intent.FLAG_ACTIVITY_CLEAR_TASK);
// Creates the PendingIntent
PendingIntent notifyPendingIntent =
        PendingIntent.getActivity(
        this,
        0,
        notifyIntent,
        PendingIntent.FLAG_UPDATE_CURRENT
);

// Puts the PendingIntent into the notification builder
builder.setContentIntent(notifyPendingIntent);
// Notifications are issued by sending them to the
// NotificationManager system service.
NotificationManager mNotificationManager =
    (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
// Builds an anonymous Notification object from the builder, and
// passes it to the NotificationManager
mNotificationManager.notify(id, builder.build());
```

Displaying Progress in a Notification

Notifications can include an animated progress indicator that shows users the status of an ongoing operation. If you can estimate how long the operation takes and how much of it is complete at any time, use the "determinate" form of the indicator (a progress bar). If you can't estimate the length of the operation, use the "indeterminate" form of the indicator (an activity indicator).

Progress indicators are displayed with the platform's implementation of the `ProgressBar`

(/reference/android/widget/ProgressBar.html) class.

To use a progress indicator on platforms starting with Android 4.0, call setProgress() (/reference/android/support/v4/app/NotificationCompat.Builder.html#setProgress(int, int, boolean)). For previous versions, you must create your own custom notification layout that includes a ProgressBar (/reference/android/widget/ProgressBar.html) view.

The following sections describe how to display progress in a notification using setProgress() (/reference/android/support/v4/app/NotificationCompat.Builder.html#setProgress(int, int, boolean)).

Displaying a fixed-duration progress indicator

To display a determinate progress bar, add the bar to your notification by calling setProgress(max, progress, false) (/reference/android/support/v4/app/NotificationCompat.Builder.html#setProgress(int, int, boolean)) and then issue the notification. As your operation proceeds, increment `progress`, and update the notification. At the end of the operation, `progress` should equal `max`. A common way to call setProgress() (/reference/android/support/v4/app/NotificationCompat.Builder.html#setProgress(int, int, boolean)) is to set `max` to 100 and then increment `progress` as a "percent complete" value for the operation.

You can either leave the progress bar showing when the operation is done, or remove it. In either case, remember to update the notification text to show that the operation is complete. To remove the progress bar, call setProgress(0, 0, false) (/reference/android/support/v4/app/NotificationCompat.Builder.html#setProgress(int, int, boolean)). For example:

```
...
mNotifyManager =
        (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
mBuilder = new NotificationCompat.Builder(this);
mBuilder.setContentTitle("Picture Download")
    .setContentText("Download in progress")
    .setSmallIcon(R.drawable.ic_notification);
// Start a lengthy operation in a background thread
new Thread(
    new Runnable() {
        @Override
        public void run() {
            int incr;
            // Do the "lengthy" operation 20 times
            for (incr = 0; incr <= 100; incr+=5) {
                    // Sets the progress indicator to a max value, the
                    // current completion percentage, and "determinate"
                    // state
                    mBuilder.setProgress(100, incr, false);
                    // Displays the progress bar for the first time.
                    mNotifyManager.notify(0, mBuilder.build());
                        // Sleeps the thread, simulating an operation
                        // that takes time
                        try {
                            // Sleep for 5 seconds
                            Thread.sleep(5*1000);
                        } catch (InterruptedException e) {
                            Log.d(TAG, "sleep failure");
                        }
            }
            // When the loop is finished, updates the notification
            mBuilder.setContentText("Download complete")
            // Removes the progress bar
                    .setProgress(0,0,false);
            mNotifyManager.notify(ID, mBuilder.build());
        }
    }
```

```
// Starts the thread by calling the run() method in its Runnable
).start();
```

Displaying a continuing activity indicator

To display an indeterminate activity indicator, add it to your notification with setProgress(0, 0, true) (/reference/android/support/v4/app/NotificationCompat.Builder.html#setProgress(int, int, boolean)) (the first two arguments are ignored), and issue the notification. The result is an indicator that has the same style as a progress bar, except that its animation is ongoing.

Issue the notification at the beginning of the operation. The animation will run until you modify your notification. When the operation is done, call setProgress(0, 0, false) (/reference/android/support/v4/app/NotificationCompat.Builder.html#setProgress(int, int, boolean)) and then update the notification to remove the activity indicator. Always do this; otherwise, the animation will run even when the operation is complete. Also remember to change the notification text to indicate that the operation is complete.

To see how activity indicators work, refer to the preceding snippet. Locate the following lines:

```
// Sets the progress indicator to a max value, the current completion
// percentage, and "determinate" state
mBuilder.setProgress(100, incr, false);
// Issues the notification
mNotifyManager.notify(0, mBuilder.build());
```

Replace the lines you've found with the following lines:

```
 // Sets an activity indicator for an operation of indeterminate length
mBuilder.setProgress(0, 0, true);
// Issues the notification
mNotifyManager.notify(0, mBuilder.build());
```

Notification Metadata

Notifications may be sorted according to metadata that you assign with the following NotificationCompat.Builder (/reference/android/support/v4/app/NotificationCompat.Builder.html) methods:

- setCategory() tells the system how to handle your app notifications when the device is in Priority mode (for example, if your notification represents an incoming call, instant message, or alarm).
- setPriority() causes notifications with the priority field set to PRIORITY_MAX or PRIORITY_HIGH to appear in a small floating window if the notification also has sound or vibration.
- addPerson() allows you to add a list of people to a notification. Your app can use this to signal to the system that it should group together notifications from the specified people, or rank notifications from these people as being more important.

Heads-up Notifications

With Android 5.0 (API level 21), notifications can appear in a small floating window (also called a *heads-up notification*) when the device is active (that is, the device is unlocked and its screen is on). These notifications appear similar to the compact form of your notification, except that the heads-up notification also shows action buttons. Users can act on, or dismiss, a heads-up notification without leaving the current app.

Examples of conditions that may trigger heads-up notifications include:

- The user's activity is in fullscreen mode (the app uses fullScreenIntent), or
- The notification has high priority and uses ringtones or vibrations

appear on the lock screen. Your app can use this functionality to provide media playback control and other common actions. Users can choose via Settings whether to display notifications on the lock screen, and you can designate whether a notification from your app is visible on the lock screen.

Setting Visibility

Your app can control the level of detail visible in notifications displayed on the lock screen. Use `setVisibility()` and specify one of the following values:

- `VISIBILITY_PUBLIC` shows the notification's full content.
- `VISIBILITY_SECRET` doesn't show any part of this notification on the lock screen.
- `VISIBILITY_PRIVATE` shows basic information, such as the notification's icon and the content title, but hides the notification's full content.

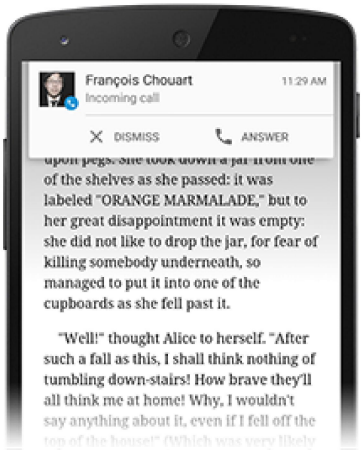

Figure 3. Fullscreen activity showing a heads-up notification

When `VISIBILITY_PRIVATE` is set, you can provide an alternate version of the notification content which hides certain details. For example, an SMS app might display a notification that shows *You have 3 new text messages*, but hides the message details. To provide this alternative notification, first create the replacement notification using `NotificationCompat.Builder`. When you have the object you want, attach the replacement notification to it through the `setPublicVersion()` method.

Controlling Media Playback on the Lock Screen

In Android 5.0 (API level 21) the lock screen no longer displays media controls based on the `RemoteControlClient` (/reference/android/media/RemoteControlClient.html), which is now deprecated. Instead, use the `Notification.MediaStyle` (/reference/android/app/Notification.MediaStyle.html) template with the `addAction()` (/reference/android/app/Notification.Builder.html#addAction(android.app.Notification.Action)) method, which converts actions into clickable icons.

> Note: The template and the `addAction()` (/reference/android/app/Notification.Builder.html#addAction(android.app.Notification.Action)) method are not included in the support library, so these features run in Android 5.0 and higher only.

To display media playback controls on the lock screen in Android 5.0, set the visibility to `VISIBILITY_PUBLIC` (/reference/android/support/v4/app/NotificationCompat.html#VISIBILITY_PUBLIC), as described above. Then add the actions and set the `Notification.MediaStyle` (/reference/android/app/Notification.MediaStyle.html) template, as described in the following sample code:

```
Notification notification = new Notification.Builder(context)
    // Show controls on lock screen even when user hides sensitive content.
    .setVisibility(Notification.VISIBILITY_PUBLIC)
    .setSmallIcon(R.drawable.ic_stat_player)
    // Add media control buttons that invoke intents in your media service
    .addAction(R.drawable.ic_prev, "Previous", prevPendingIntent) // #0
    .addAction(R.drawable.ic_pause, "Pause", pausePendingIntent)  // #1
    .addAction(R.drawable.ic_next, "Next", nextPendingIntent)     // #2
    // Apply the media style template
    .setStyle(new Notification.MediaStyle()
```

```
        .setShowActionsInCompactView(1 /* #1: pause button */)
        .setMediaSession(mMediaSession.getSessionToken())
        .setContentTitle("Wonderful music")
        .setContentText("My Awesome Band")
        .setLargeIcon(albumArtBitmap)
        .build();
```

> Note: The deprecation of RemoteControlClient (/reference/android/media/RemoteControlClient.html) has further implications for controlling media. See Media Playback Control (/about/versions/android-5.0.html#MediaPlaybackControl) for more information about the new APIs for managing the media session and controlling playback.

Custom Notification Layouts

The notifications framework allows you to define a custom notification layout, which defines the notification's appearance in a RemoteViews (/reference/android/widget/RemoteViews.html) object. Custom layout notifications are similar to normal notifications, but they're based on a RemoteViews (/reference/android/widget/RemoteViews.html) defined in a XML layout file.

The height available for a custom notification layout depends on the notification view. Normal view layouts are limited to 64 dp, and expanded view layouts are limited to 256 dp.

To define a custom notification layout, start by instantiating a RemoteViews (/reference/android/widget/RemoteViews.html) object that inflates an XML layout file. Then, instead of calling methods such as setContentTitle() (/reference/android/support/v4/app/NotificationCompat.Builder.html#setContentTitle(java.lang.CharSequence)), call setContent() (/reference/android/support/v4/app/NotificationCompat.Builder.html#setContent(android.widget.RemoteViews)). To set content details in the custom notification, use the methods in RemoteViews (/reference/android/widget/RemoteViews.html) to set the values of the view's children:

1. Create an XML layout for the notification in a separate file. You can use any file name you wish, but you must use the extension .xml
2. In your app, use RemoteViews methods to define your notification's icons and text. Put this RemoteViews object into your NotificationCompat.Builder by calling setContent(). Avoid setting a background Drawable on your RemoteViews object, because your text color may become unreadable.

The RemoteViews (/reference/android/widget/RemoteViews.html) class also includes methods that you can use to easily add a Chronometer (/reference/android/widget/Chronometer.html) or ProgressBar (/reference/android/widget/ProgressBar.html) to your notification's layout. For more information about creating custom layouts for your notification, refer to the RemoteViews (/reference/android/widget/RemoteViews.html) reference documentation.

> Caution: When you use a custom notification layout, take special care to ensure that your custom layout works with different device orientations and resolutions. While this advice applies to all View layouts, it's especially important for notifications because the space in the notification drawer is very restricted. Don't make your custom layout too complex, and be sure to test it in various configurations.

Using style resources for custom notification text

Always use style resources for the text of a custom notification. The background color of the notification can vary across different devices and versions, and using style resources helps you account for this. Starting in Android 2.3, the system defined a style for the standard notification layout text. If you use the same style in applications that target Android 2.3 or higher, you'll ensure that your text is visible against the display background.