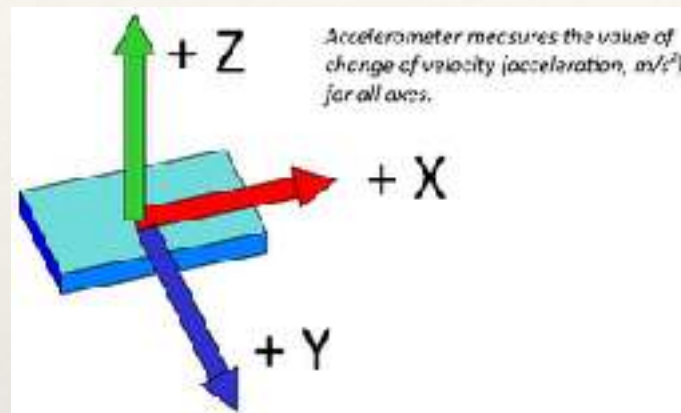


Introduction to Mobile Programming

Android Programming

Chapter 4

Sensors Overview



1. The Android sensor framework lets you access many types of sensors. Some of these sensors are hardware-based and some are software-based.
 1. Determine which sensors are available on a device.
 2. Determine an individual sensor's capabilities, such as its maximum range, manufacturer, power requirements, and resolution.
 3. Acquire raw sensor data and define the minimum rate at which you acquire sensor data.
 4. Register and unregister sensor event listeners that monitor sensor changes.

Sensors



Sensor	Type	Description	Common Uses
TYPE_ACCELEROMETER	Hardware	Measures the acceleration force in m/s^2 that is applied to a device on all three physical axes (x, y, and z), including the force of gravity.	Motion detection (shake, tilt, etc.)
TYPE_AMBIENT_TEMPERATURE	Hardware	Measures the ambient room temperature in degrees Celsius ($^{\circ}C$). See note below.	Monitoring air temperatures.
TYPE_GRAVITY	Software or Hardware	Measures the force of gravity in m/s^2 that is applied to a device on all three physical axes (x, y, z).	Motion detection (shake, tilt, etc.)
TYPE_GYROSCOPE	Hardware	Measures a device's rate of rotation in rad/s around each of the three physical axes (x, y, and z).	Rotation detection (spin, turn, etc.)
TYPE_LIGHT	Hardware	Measures the ambient light level (illumination) in lx.	Controlling screen brightness.
TYPE_LINEAR_ACCELERATION	Software or Hardware	Measures the acceleration force in m/s^2 that is applied to a device on all three physical axes (x, y, and z), excluding the force of gravity.	Monitoring acceleration along a single axis.
TYPE_MAGNETIC_FIELD	Hardware	Measures the ambient geomagnetic field for all three physical axes (x, y, z) in μT .	Creating a compass.
TYPE_ORIENTATION	Software	Measures degrees of rotation that a device makes around all three physical axes (x, y, z). As of API level 3 you can obtain the inclination matrix and rotation matrix for a device by using the gravity sensor and the geomagnetic field sensor in conjunction with the getRotationMatrix() method.	Determining device position.
TYPE_PRESSURE	Hardware	Measures the ambient air pressure in hPa or mbar.	Monitoring air pressure changes.
TYPE_PROXIMITY	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.
TYPE_RELATIVE_HUMIDITY	Hardware	Measures the relative ambient humidity in percent (%).	Monitoring dewpoint, absolute, and relative humidity.
TYPE_ROTATION_VECTOR	Software or Hardware	Measures the orientation of a device by providing the three elements of the device's rotation vector.	Motion detection and rotation detection.
TYPE_TEMPERATURE	Hardware	Measures the temperature of the device in degrees Celsius ($^{\circ}C$). This sensor implementation varies across devices and this sensor was replaced with the TYPE_AMBIENT_TEMPERATURE sensor in API Level 14.	Monitoring temperatures.

Sensor Framework

1. **SensorManager**

- a. This class provides various methods for accessing and listing sensors, registering and unregistering sensor event listeners, and acquiring orientation information. This class also provides several sensor constants that are used to report sensor accuracy, set data acquisition rates, and calibrate sensors.

2. **Sensor**

- a. This class provides various methods that let you determine a sensor's capabilities.

3. **SensorEvent**

- a. A sensor event object includes the following information: the raw sensor data, the type of sensor that generated the event, the accuracy of the data, and the timestamp for the event.

4. **SensorEventListener**

- 1. You can use this interface to create two callback methods that receive notifications (sensor events) when sensor values change or when sensor accuracy changes.

Identifying Sensors and Sensor Capabilities

```
private SensorManager sensorManager;  
...  
sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
```

```
List<Sensor> deviceSensors = sensorManager.getSensorList(Sensor.TYPE_ALL);
```

```
private SensorManager sensorManager;  
...  
sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);  
if (sensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD) != null){  
    // Success! There's a magnetometer.  
} else {  
    // Failure! No magnetometer.  
}
```

`getResolution()`

`getMaximumRange()`

`getPower()`

`getVendor()`

`getVersion()`

Available Sensors

```
private SensorManager sensorManager;
private Sensor mSensor;

...

sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
mSensor = null;

if (sensorManager.getDefaultSensor(Sensor.TYPE_GRAVITY) != null){
    List<Sensor> gravSensors = sensorManager.getSensorList(Sensor.TYPE_GRAVITY);
    for(int i=0; i<gravSensors.size(); i++) {
        if ((gravSensors.get(i).getVendor().contains("Google LLC")) &&
            (gravSensors.get(i).getVersion() == 3)){
            // Use the version 3 gravity sensor.
            mSensor = gravSensors.get(i);
        }
    }
}
if (mSensor == null){
    // Use the accelerometer.
    if (sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER) != null){
        mSensor = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
    } else{
        // Sorry, there are no accelerometers on your device.
        // You can't play this game.
    }
}
```

onSensorChanged() - onAccuracyChanged()

```
public class SensorActivity extends Activity implements SensorEventListener {
    private SensorManager sensorManager;
    private Sensor mLight;

    @Override
    public final void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        mLight = sensorManager.getDefaultSensor(Sensor.TYPE_LIGHT);
    }

    @Override
    public final void onAccuracyChanged(Sensor sensor, int accuracy) {
        // Do something here if sensor accuracy changes.
    }

    @Override
    public final void onSensorChanged(SensorEvent event) {
        // The light sensor returns a single value.
        // Many sensors return 3 values, one for each axis.
        float lux = event.values[0];
        // Do something with this sensor value.
    }

    @Override
    protected void onResume() {
        super.onResume();
        sensorManager.registerListener(this, mLight, SensorManager.SENSOR_DELAY_NORMAL);
    }

    @Override
    protected void onPause() {
        super.onPause();
        sensorManager.unregisterListener(this);
    }
}
```

Google Play Filters

```
<uses-feature android:name="android.hardware.sensor.accelerometer"  
             android:required="true" />
```

Reporting Modes - Android 9

1. Continuous









- a. Events are generated at a constant rate defined by the `sampling_period_ns` parameter passed to the `batch` function. Example sensors using the continuous reporting mode are `accelerometers` and `gyroscopes`.

2. On-change

- a. Events are generated only if the measured values have changed. Example sensors using the on-change reporting mode are the step counter, proximity, and heart rate sensor types.

3. One-shot

- a. Upon detection of an event, the sensor deactivates itself and then sends a single event through the HAL. One-shot sensors are sometimes referred to as trigger sensors.

Sensor type	Category	Underlying physical sensors	Reporting mode
Game rotation vector	Attitude	Accelerometer, gyroscope, MUST NOT USE magnetometer	Continuous
Geomagnetic rotation vector  %	Attitude	Accelerometer, magnetometer, MUST NOT USE gyroscope	Continuous
Glance gesture  %	Interaction	Undefined	One-shot
Gravity	Attitude	Accelerometer, gyroscope	Continuous
Gyroscope uncalibrated	Uncalibrated	Gyroscope	Continuous
Linear acceleration	Activity	Accelerometer, gyroscope (if present), or magnetometer (if gyro not present)	Continuous
Magnetic field uncalibrated	Uncalibrated	Magnetometer	Continuous
Orientation (deprecated)	Attitude	Accelerometer, magnetometer, gyroscope (if present)	Continuous
Pick up gesture  %	Interaction	Undefined	One-shot
Rotation vector	Attitude	Accelerometer, magnetometer, gyroscope	Continuous
Significant motion  %	Activity	Accelerometer (or another as long as very low power)	One-shot
Step counter  %	Activity	Accelerometer	On-change
Step detector  %	Activity	Accelerometer	Special
Tilt detector  %	Activity	Accelerometer	Special
Wake up gesture  %	Interaction	Undefined	One-shot

Motion Sensors

1. All of the motion sensors return multi-dimensional arrays of sensor values for each `SensorEvent`.

`TYPE_ACCELEROMETER`

<code>SensorEvent.values[0]</code>	Acceleration force along the x axis (including gravity).	m/s^2
<code>SensorEvent.values[1]</code>	Acceleration force along the y axis (including gravity).	
<code>SensorEvent.values[2]</code>	Acceleration force along the z axis (including gravity).	

```
private SensorManager sensorManager;
private Sensor sensor;
private TriggerEventListener triggerEventListener;
...
sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
sensor = sensorManager.getDefaultSensor(Sensor.TYPE_SIGNIFICANT_MOTION);

triggerEventListener = new TriggerEventListener() {
    @Override
    public void onTrigger(TriggerEvent event) {
        // Do work
    }
};

sensorManager.requestTriggerSensor(triggerEventListener, mSensor);
```

Position Sensors

1. Position sensors are useful for determining a device's physical position in the world's frame of reference

TYPE_PROXIMITY

SensorEvent.values[0] Distance from object.²

cm

- **Azimuth (degrees of rotation about the -z axis).** This is the angle between the device's current compass direction and magnetic north. If the top edge of the device faces magnetic north, the azimuth is 0 degrees; if the top edge faces south, the azimuth is 180 degrees. Similarly, if the top edge faces east, the azimuth is 90 degrees, and if the top edge faces west, the azimuth is 270 degrees.
- **Pitch (degrees of rotation about the x axis).** This is the angle between a plane parallel to the device's screen and a plane parallel to the ground. If you hold the device parallel to the ground with the bottom edge closest to you and tilt the top edge of the device toward the ground, the pitch angle becomes positive. Tilting in the opposite direction — moving the top edge of the device away from the ground — causes the pitch angle to become negative. The range of values is -180 degrees to 180 degrees.
- **Roll (degrees of rotation about the y axis).** This is the angle between a plane perpendicular to the device's screen and a plane perpendicular to the ground. If you hold the device parallel to the ground with the bottom edge closest to you and tilt the left edge of the device toward the ground, the roll angle becomes positive. Tilting in the opposite direction — moving the right edge of the device toward the ground — causes the roll angle to become negative. The range of values is -90 degrees to 90 degrees.

Environment Sensors

Sensor	Sensor event data	Units of measure	Data description
TYPE_AMBIENT_TEMPERATURE	<code>event.values[0]</code>	°C	Ambient air temperature.
TYPE_LIGHT	<code>event.values[0]</code>	lx	Illuminance.
TYPE_PRESSURE	<code>event.values[0]</code>	hPa or mbar	Ambient air pressure.
TYPE_RELATIVE_HUMIDITY	<code>event.values[0]</code>	%	Ambient relative humidity.
TYPE_TEMPERATURE	<code>event.values[0]</code>	°C	Device temperature. ¹

Sensor	Sensor event data	Description	Units of measure
TYPE_ACCELEROMETER	SensorEvent.values[0]	Acceleration force along the x axis (including gravity)	m/s ²
	SensorEvent.values[1]	Acceleration force along the y axis (including gravity).	
	SensorEvent.values[2]	Acceleration force along the z axis (including gravity).	
TYPE_ACCELEROMETER_UNCALIBRATED	SensorEvent.values[0]	Measured acceleration along the X axis without any bias compensation.	m/s ²
	SensorEvent.values[1]	Measured acceleration along the Y axis without any bias compensation.	
	SensorEvent.values[2]	Measured acceleration along the Z axis without any bias compensation.	
	SensorEvent.values[3]	Measured acceleration along the X axis with estimated bias compensation.	
	SensorEvent.values[4]	Measured acceleration along the Y axis with estimated bias compensation.	
	SensorEvent.values[5]	Measured acceleration along the Z axis with estimated bias compensation.	
TYPE_GRAVITY	SensorEvent.values[0]	Force of gravity along the x axis.	m/s ²
	SensorEvent.values[1]	Force of gravity along the y axis.	
	SensorEvent.values[2]	Force of gravity along the z axis.	

TYPE_GYROSCOPE	SensorEvent. values[0]	Rate of rotation around the x axis.	rad/s
	SensorEvent. values[1]	Rate of rotation around the y axis.	
	SensorEvent. values[2]	Rate of rotation around the z axis.	
TYPE_GYROSCOPE_UNCALIBRATED	SensorEvent. values[0]	Rate of rotation (without drift compensation) around the x axis.	rad/s
	SensorEvent. values[1]	Rate of rotation (without drift compensation) around the y axis.	
	SensorEvent. values[2]	Rate of rotation (without drift compensation) around the z axis.	
	SensorEvent. values[3]	Estimated drift around the x axis.	
	SensorEvent. values[4]	Estimated drift around the y axis.	
	SensorEvent. values[5]	Estimated drift around the z axis.	
TYPE_LINEAR_ACCELERATION	SensorEvent. values[0]	Acceleration force along the x axis (excluding gravity).	m/s ²
	SensorEvent. values[1]	Acceleration force along the y axis (excluding gravity).	
	SensorEvent. values[2]	Acceleration force along the z axis (excluding gravity).	
TYPE_ROTATION_VECTOR	SensorEvent. values[0]	Rotation vector component along the x axis ($x * \sin(\theta/2)$).	Unitless
	SensorEvent. values[1]	Rotation vector component along the y axis ($y * \sin(\theta/2)$).	
	SensorEvent. values[2]	Rotation vector component along the z axis ($z * \sin(\theta/2)$).	
	SensorEvent. values[3]	Scalar component of the rotation vector ($(\cos(\theta/2))$). ¹	
TYPE_SIGNIFICANT_MOTION	N/A	N/A	N/A
TYPE_STEP_COUNTER	SensorEvent. values[0]	Number of steps taken by the user since the last reboot while the sensor was activated.	Steps
TYPE_STEP_DETECTOR	N/A	N/A	N/A

Sensor	Sensor event data	Description	Units of measure
TYPE_GAME_ROTATION_VECTOR	SensorEvent. values[0]	Rotation vector component along the x axis ($x * \sin(\theta/2)$).	Unitless
	SensorEvent. values[1]	Rotation vector component along the y axis ($y * \sin(\theta/2)$).	
	SensorEvent. values[2]	Rotation vector component along the z axis ($z * \sin(\theta/2)$).	
TYPE_GEOMAGNETIC_ROTATION_VECTOR	SensorEvent. values[0]	Rotation vector component along the x axis ($x * \sin(\theta/2)$).	Unitless
	SensorEvent. values[1]	Rotation vector component along the y axis ($y * \sin(\theta/2)$).	
	SensorEvent. values[2]	Rotation vector component along the z axis ($z * \sin(\theta/2)$).	
TYPE_MAGNETIC_FIELD	SensorEvent. values[0]	Geomagnetic field strength along the x axis.	μT
	SensorEvent. values[1]	Geomagnetic field strength along the y axis.	
	SensorEvent. values[2]	Geomagnetic field strength along the z axis.	
TYPE_MAGNETIC_FIELD_UNCALIBRATED	SensorEvent. values[0]	Geomagnetic field strength (without hard iron calibration) along the x axis.	μT
	SensorEvent. values[1]	Geomagnetic field strength (without hard iron calibration) along the y axis.	
	SensorEvent. values[2]	Geomagnetic field strength (without hard iron calibration) along the z axis.	
	SensorEvent. values[3]	Iron bias estimation along the x axis.	
	SensorEvent. values[4]	Iron bias estimation along the y axis.	
	SensorEvent. values[5]	Iron bias estimation along the z axis.	
TYPE_ORIENTATION ¹	SensorEvent. values[0]	Azimuth (angle around the z-axis).	Degrees
	SensorEvent. values[1]	Pitch (angle around the x-axis).	
	SensorEvent. values[2]	Roll (angle around the y-axis).	
TYPE_PROXIMITY	SensorEvent. values[0]	Distance from object. ²	cm

Sensor-Rate Limiting

To protect potentially sensitive information about users, if your app targets Android 12 (API level 31) or higher, the system places a limit on the refresh rate of data from certain motion sensors and position sensors. This data includes values recorded by the device's [accelerometer](#), [gyroscope](#), and [geomagnetic field sensor](#).

The refresh rate limit depends on how you access sensor data:

- If you call the `registerListener()` method to [monitor sensor events](#), the sensor sampling rate is limited to 200 Hz. This is true for all overloaded variants of the `registerListener()` method.
- If you use the `SensorDirectChannel` class, the sensor sampling rate is limited to `RATE_NORMAL`, which is usually about 50 Hz.

If your app needs to gather motion sensor data at a higher rate, you must declare the `HIGH_SAMPLING_RATE_SENSORS` permission, as shown in the following code snippet. Otherwise, if your app tries to gather motion sensor data at a higher rate without declaring this permission, a `SecurityException` occurs.

```
<manifest ...>
    <uses-permission android:name="android.permission.HIGH_SAMPLING_RATE_SENSORS"/>
    <application ...>
        ...
    </application>
</manifest>
```


Notifications Overview

1. A notification is a message that Android displays outside your app's UI to provide the user with reminders, communication from other people, or other timely information from your app.
2. Users can tap the notification to open your app or take an action directly from the notification.

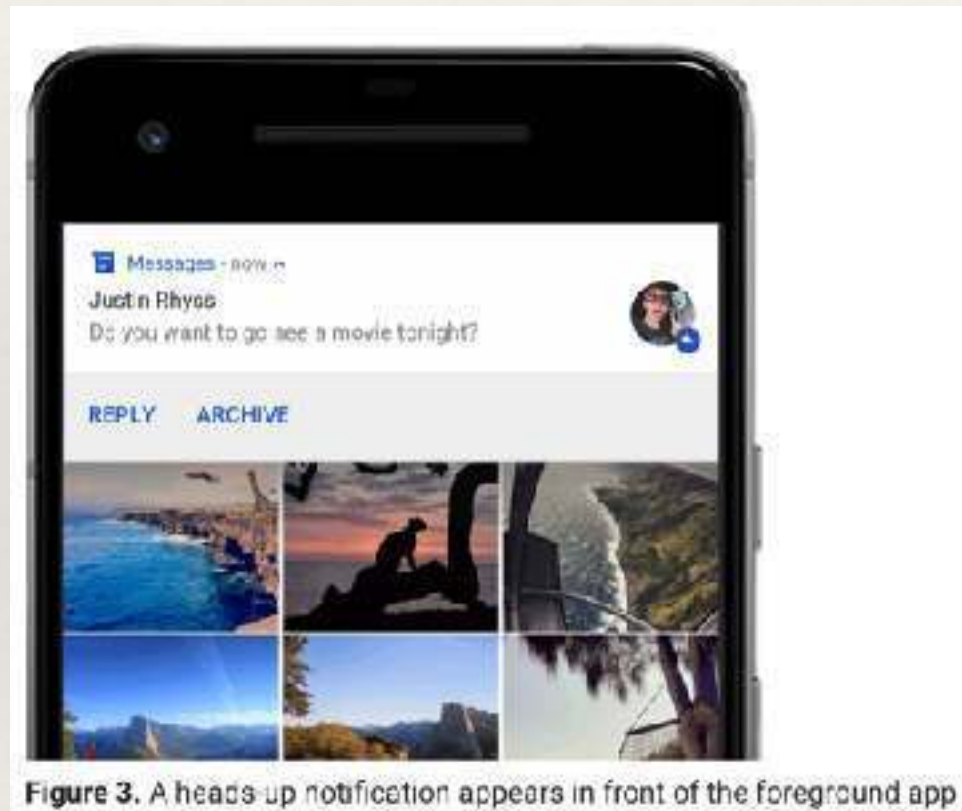


Figure 1. Notification icons appear on the left side of the status bar

Notification Anatomy

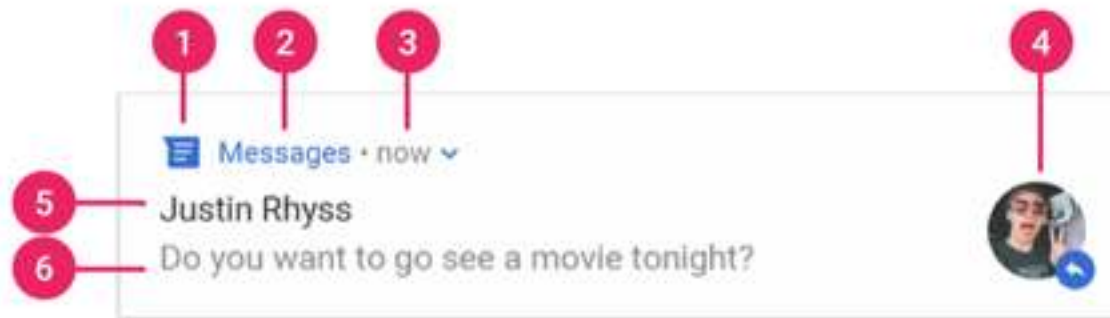
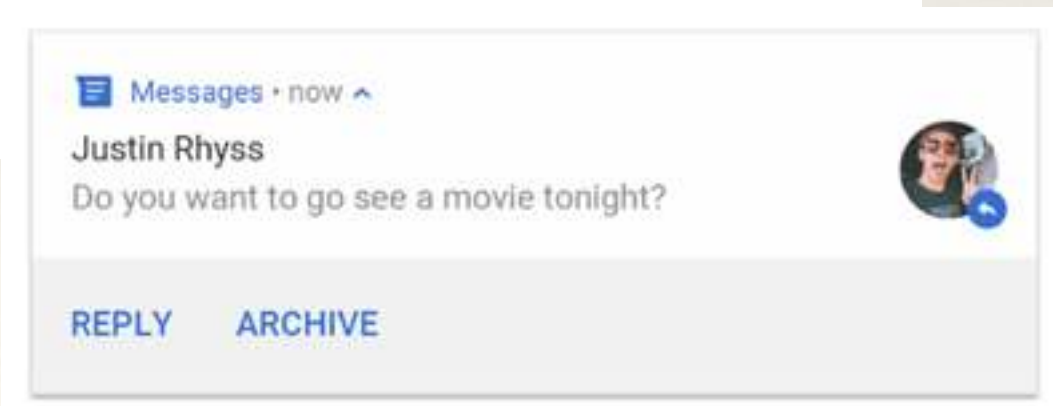


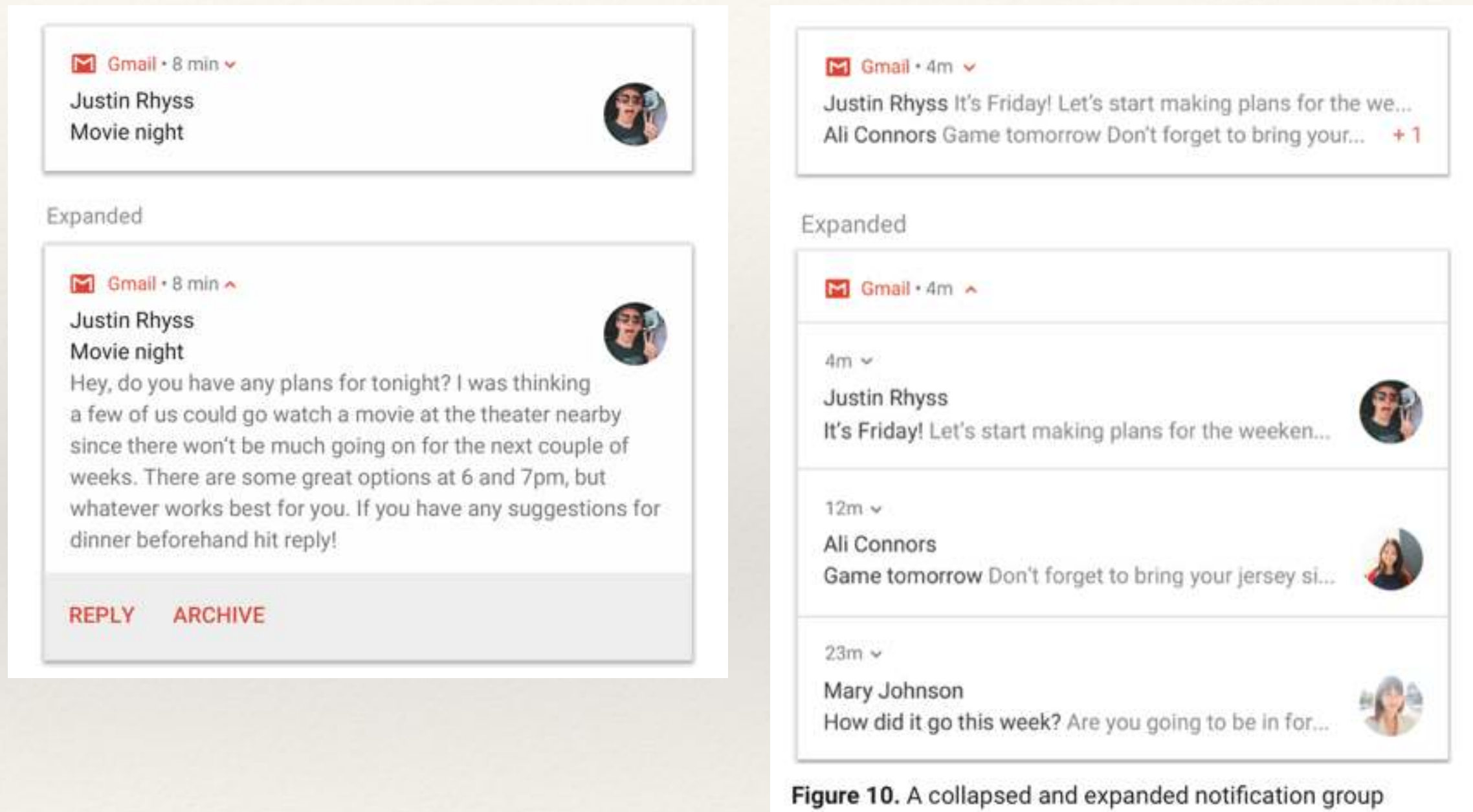
Figure 7. A notification with basic details

The most common parts of a notification are indicated in figure 7 as follows:

- 1 Small icon: This is required and set with `setSmallIcon()`.
- 2 App name: This is provided by the system.
- 3 Time stamp: This is provided by the system but you can override with `setWhen()` or hide it with `setShowWhen(false)`.
- 4 Large icon: This is optional (usually used only for contact photos; do not use it for your app icon) and set with `setLargeIcon()`.
- 5 Title: This is optional and set with `setContentTitle()`.
- 6 Text: This is optional and set with `setContentText()`.



Expandable Notification



Expandable Notification

```
Notification notification = new NotificationCompat.Builder(context, CHANNEL_ID)
    .setSmallIcon(R.drawable.new_mail)
    .setContentTitle(emailObject.getSenderName())
    .setContentText(emailObject.getSubject())
    .setLargeIcon(emailObject.getSenderAvatar())
    .setStyle(new NotificationCompat.BigTextStyle()
        .bigText(emailObject.getSubjectAndSnippet()))
    .build();
```

```
Notification notification = new NotificationCompat.Builder(context, CHANNEL_ID)
    .setSmallIcon(R.drawable.new_post)
    .setContentTitle(imageTitle)
    .setContentText(imageDescription)
    .setLargeIcon(myBitmap)
    .setStyle(new NotificationCompat.BigPictureStyle()
        .bigPicture(myBitmap)
        .bigLargeIcon(null))
    .build();
```

Setting the Notification's Tap Action

```
// Create an explicit intent for an Activity in your app
Intent intent = new Intent(this, AlertDetails.class);
intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TASK);
PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, intent, PendingIntent.FLAG_IMMUTABLE);

NotificationCompat.Builder builder = new NotificationCompat.Builder(this, CHANNEL_ID)
    .setSmallIcon(R.drawable.notification_icon)
    .setContentTitle("My notification")
    .setContentText("Hello World!")
    .setPriority(NotificationCompat.PRIORITY_DEFAULT)
    // Set the intent that will fire when the user taps the notification
    .setContentIntent(pendingIntent)
    .setAutoCancel(true);
```

```
NotificationManagerCompat notificationManager = NotificationManagerCompat.from(this);

// notificationId is a unique int for each notification that you must define
notificationManager.notify(notificationId, builder.build());
```


Adding Action Buttons



```
Intent snoozeIntent = new Intent(this, MyBroadcastReceiver.class);
snoozeIntent.setAction(ACTION_SNOOZE);
snoozeIntent.putExtra(EXTRA_NOTIFICATION_ID, 0);
PendingIntent snoozePendingIntent =
    PendingIntent.getBroadcast(this, 0, snoozeIntent, 0);

NotificationCompat.Builder builder = new NotificationCompat.Builder(this, CHANNEL_ID)
    .setSmallIcon(R.drawable.notification_icon)
    .setContentTitle("My notification")
    .setContentText("Hello World!")
    .setPriority(NotificationCompat.PRIORITY_DEFAULT)
    .setContentIntent(pendingIntent)
    .addAction(R.drawable.ic_snooze, getString(R.string.snooze),
        snoozePendingIntent);
```

Setting Lock-Screen Visibility

To control the level of detail visible in the notification from the lock screen, call `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.

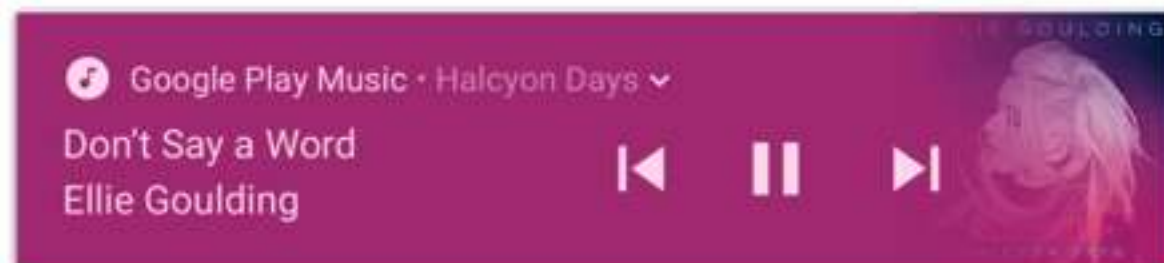
When `VISIBILITY_PRIVATE` is set, you can also provide an alternate version of the notification content which hides certain details. For example, an SMS app might display a notification that shows *You have 3 new text messages*, but hides the message contents and senders. To provide this alternative notification, first create the alternative notification with `NotificationCompat.Builder` as usual. Then attach the alternative notification to the normal notification with `setPublicVersion()`.

However, the user always has final control over whether their notifications are visible on the lock screen and can even control that based on your app's notification channels.

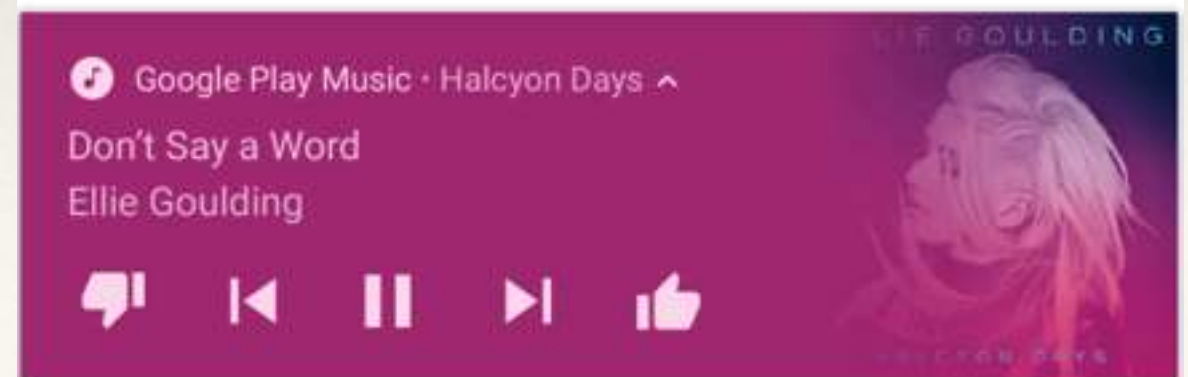
Create a Notification with MediaControls

```
Notification notification = new NotificationCompat.Builder(context, CHANNEL_ID)
    // Show controls on lock screen even when user hides sensitive content.
    .setVisibility(NotificationCompat.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 android.support.v4.media.app.Notification.MediaStyle()
        .setShowActionsInCompactView(1 /* #1: pause button */)
        .setMediaSession(mediaSession.getSessionToken()))
    .setContentTitle("Wonderful music")
    .setContentText("My Awesome Band")
    .setLargeIcon(albumArtBitmap)
    .build();
```

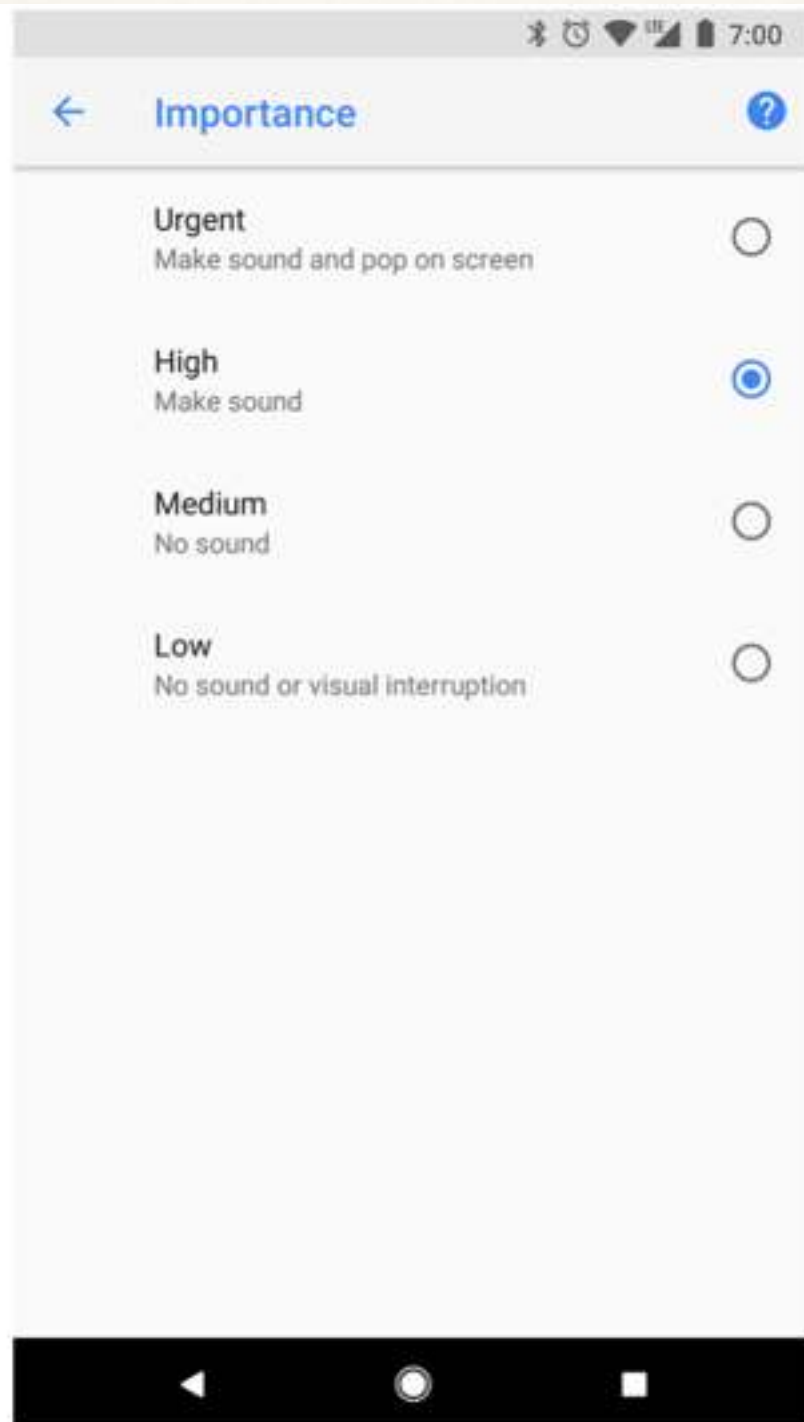
Collapsed



Expanded



Notification Channels and Importance



The possible importance levels are the following:

- Urgent: Makes a sound and appears as a heads-up notification.
- High: Makes a sound.
- Medium: No sound.
- Low: No sound and does not appear in the status bar.

Figure 12. Users can change the importance of each channel on Android 8.0 and higher

Create a Basic Notification

```
NotificationCompat.Builder builder = new NotificationCompat.Builder(this, CHANNEL_ID)
    .setSmallIcon(R.drawable.notification_icon)
    .setContentTitle(textTitle)
    .setContentText(textContent)
    .setPriority(NotificationCompat.PRIORITY_DEFAULT);
```

```
NotificationCompat.Builder builder = new NotificationCompat.Builder(this, CHANNEL_ID)
    .setSmallIcon(R.drawable.notification_icon)
    .setContentTitle("My notification")
    .setContentText("Much longer text that cannot fit one line...")
    .setStyle(new NotificationCompat.BigTextStyle()
        .bigText("Much longer text that cannot fit one line..."))
    .setPriority(NotificationCompat.PRIORITY_DEFAULT);
```

```
// Create an explicit intent for an Activity in your app
Intent intent = new Intent(this, AlertDetails.class);
intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TASK);
PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, intent, 0);

NotificationCompat.Builder builder = new NotificationCompat.Builder(this, CHANNEL_ID)
    .setSmallIcon(R.drawable.notification_icon)
    .setContentTitle("My notification")
    .setContentText("Hello World!")
    .setPriority(NotificationCompat.PRIORITY_DEFAULT)
    // Set the intent that will fire when the user taps the notification
    .setContentIntent(pendingIntent)
    .setAutoCancel(true);
```

Show the Notification and Add Action Buttons

```
NotificationManagerCompat notificationManager = NotificationManagerCompat.from(this);

// notificationId is a unique int for each notification that you must define
notificationManager.notify(notificationId, builder.build());
```

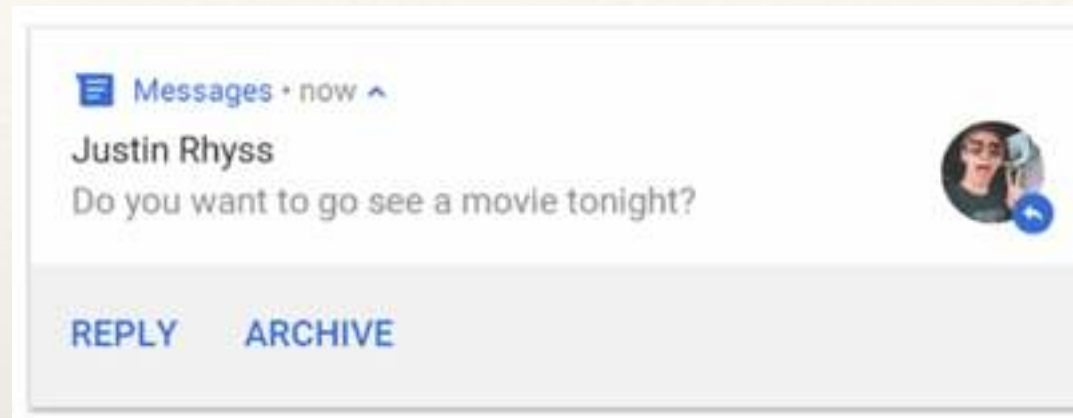


Figure 2. A notification with one action button

```
Intent snoozeIntent = new Intent(this, MyBroadcastReceiver.class);
snoozeIntent.setAction(ACTION_SNOOZE);
snoozeIntent.putExtra(EXTRA_NOTIFICATION_ID, 0);
PendingIntent snoozePendingIntent =
    PendingIntent.getBroadcast(this, 0, snoozeIntent, 0);

NotificationCompat.Builder builder = new NotificationCompat.Builder(this, CHANNEL_ID)
    .setSmallIcon(R.drawable.notification_icon)
    .setContentTitle("My notification")
    .setContentText("Hello World!")
    .setPriority(NotificationCompat.PRIORITY_DEFAULT)
    .setContentIntent(pendingIntent)
    .addAction(R.drawable.ic_snooze, getString(R.string.snooze),
        snoozePendingIntent);
```


Notification Actions



```
Notification moreSecureNotification = new Notification.Builder(  
    context, NotificationListenerVerifierActivity.TAG)  
    .addAction(...)  
  
    // This notification always requests authentication when invoked  
    // from a lock screen.  
    .setAuthenticationRequired(true)  
    .build();
```