Alerting Users via Notifications

Pop-up messages, tray icons and their associated “bubble” messages, bouncing dock icons...you are no doubt used to programs trying to get your attention, sometimes for good reason. Your phone also probably chirps at you for more than just incoming calls: low battery, alarm clocks, appointment notifications, incoming text and e-mail messages, and so on.

Not surprisingly, Android has a whole framework for dealing with these sorts of things, collectively called notifications, as described in this chapter.

Notification Configuration

A service, running in the background, needs a way to let users know something of interest has occurred, such as when e-mail has been received. Moreover, the service may need some way to steer users to an activity where they can act upon the event, such as reading a received message. For this, Android supplies status bar icons, flashing lights, and other indicators collectively known as notifications.

Your current phone may already have such icons, to indicate battery life, signal strength, whether Bluetooth is enabled, and the like. With Android, applications can add their own status bar icons, with an eye toward having them appear only when needed (e.g., a message has arrived).

In Android, you can raise notifications via the NotificationManager, which is a system service. To use it, you need to get the service object via getSystemService(NOTIFICATION_SERVICE) from your activity. The NotificationManager gives you three methods: one to raise a Notification (notify()) and two to get rid of an existing Notification (cancel() and cancelAll()).
CHAPTER 37: Alerting Users via Notifications

The `notify()` method takes a `Notification`, which is a data structure that spells out what form your pester should take. The capabilities of this object are described in the following sections.

**Hardware Notifications**

You can flash LEDs on the device by setting `lights` to `true`, also specifying the color (as an `#ARGB` value in `ledARGB`) and what pattern the light should blink in (by providing off/on durations in milliseconds for the light via `ledOnMS` and `ledOffMS`). Note, however, that Android devices will apply best efforts to meet your color request, meaning that different devices may give you different colors, or perhaps no control over color at all. For example, the Motorola CLIQ has only a white LED, so you can ask for any color you want, and you will get white. Note that you will need to OR (`|`) the `Notification.FLAG_SHOW_LIGHTS` value into the public `flags` field on the `Notification` object for flashing of the LED to work.

You can play a sound, using a `Uri` to a piece of content held, perhaps, by a `ContentManager` (`sound`). Think of this as a ringtone for your application.

You can vibrate the device, controlled via a `long[]`, indicating the on/off patterns (in milliseconds) for the vibration (`vibrate`). You might do this by default, or you might make it an option the user can choose when circumstances require a more subtle notification than a ringtone. To use this, though, you will need to request the `VIBRATE` permission (permissions are discussed in Chapter 38).

All of these options, by default, happen once (e.g., one LED flash or one playback of the sound). If you want to have them persist until the `Notification` is canceled, you will need to set the `flags` public field in your `Notification` to include `FLAG_INSISTENT`.

Instead of manually specifying the hardware options, you can also use the `defaults` field in the `Notification`, setting it to `DEFAULT_LIGHTS`, `DEFAULT_SOUND`, `DEFAULT_VIBRATE`, or `DEFAULT_ALL`, which will use platform defaults for all hardware options.

**Icons**

While the flashing lights, sounds, and vibrations are aimed at getting somebody to look at the device, icons are designed to take them the next step and tell them what’s so important.

To set up an icon for a `Notification`, you need to set two public fields: `icon`, where you provide the identifier of a `Drawable` resource representing the icon, and `contentIntent`, where you supply a `PendingIntent` to be raised when the icon is clicked. A `PendingIntent` is a wrapper around a regular `Intent` that allows the `Intent` to be invoked later, by another process, to start an activity or whatever. Typically, a `Notification` will trigger an activity, in which case you would create the `PendingIntent` via the static `getActivity()` method and give it an `Intent` that identifies one of your activities. That being said, you could have the `Notification` send a broadcast `Intent` instead, by using a `getBroadcast()` version of a `PendingIntent`. Android 4.0 has expanded the variety of...