Using Alarms

Alarms are an application independent way of fi ring Intents at predetermined times.

Alarms are set outside the scope of your applications, so they can be used to trigger application events or actions even after your application has been closed. They can be particularly powerful in combination with Broadcast Receivers, allowing you to set Alarms that launch applications or perform actions without applications needing to be open and active until they're required.

For example, you can use Alarms to implement an alarm clock application, perform regular network lookups, or schedule time-consuming or cost-bound operations at "off peak" times.

For timing operations that occur only during the lifetime of your applications, the Handler *class in combination with Timers and Threads is a better approach as it allows Android better control over system resources.*

Alarms in Android remain active while the device is in sleep mode and can optionally be set to wake the device; however, all Alarms are canceled whenever the device is rebooted.

Alarm operations are handled through the AlarmManager, a system Service accessed via getSystemService as shown below:
AlarmManager alarms = (AlarmManager)getSystemService(Context.ALARM_SERVICE);

To create a new Alarm, use the set method and specify an alarm type, trigger time, and a Pending Intent to fi re when the Alarm triggers. If the Alarm you set occurs in the past, it will be triggered immediately.

There are four alarm types available. Your selection will determine if the time value passed in the set method represents a specifi c time or an elapsed wait:

- **RTC_WAKEUP** Wakes up the device to fi re the Intent at the clock time specifi ed when setting the Alarm.
- **RTC** Will fi re the Intent at an explicit time, but will not wake the device.
- **ELAPSED_REALTIME** The Intent will be fi red based on the amount of time elapsed since the device was booted, but will not wake the device. The elapsed time includes any period of time the device was asleep. Note that the time elapsed is since it was last booted.
- **ELAPSED_REALTIME_WAKEUP** Will wake up the device if necessary and fi re the Intent after a specifi ed length of time has passed since the device was booted.

The Alarm creation process is demonstrated in the snippet below:

```
int alarmType = AlarmManager.ELAPSED_REALTIME_WAKEUP;
long timeOrLengthofWait = 10000;
String ALARM_ACTION = "ALARM_ACTION";
Intent intentToFire = new Intent(ALARM_ACTION);
PendingIntent pendingIntent = PendingIntent.getBroadcast(this, 0, intentToFire, 0);
alarms.set(alarmType, timeOrLengthofWait, pendingIntent);
```

When the Alarm goes off, the Pending Intent you specifi ed will be fi red. Setting a second Alarm using the same Pending Intent replaces the preexisting Alarm.

To cancel an Alarm, call cancel on the Alarm Manager, passing in the Pending Intent you no longer wish to trigger, as shown in the snippet below:

```
alarms.cancel(pendingIntent);
```

In the following code snippet, two Alarms are set and the fi rst one is subsequently canceled. The fi rst is explicitly set to a specifi c time and will wake up the device in order to fi re. The second is set for 30 minutes of time elapsed since the device was started, but will not wake the device if it's sleeping.

```
AlarmManager alarms = (AlarmManager)getSystemService(Context.ALARM_SERVICE);
String MY_RTC_ALARM = "MY_RTC_ALARM";
String ALARM_ACTION = "MY_ELAPSED_ALARM";
PendingIntent rtcIntent = PendingIntent.getBroadcast(this, 0,
new Intent(MY_RTC_ALARM), 1);
```

```
PendingIntent elapsedIntent = PendingIntent.getBroadcast(this, 0, new Intent(ALARM_ACTION), 1);
// Wakeup and fire intent in 5 hours.
Date t = new Date();
t.setTime(java.lang.System.currentTimeMillis() + 60*1000*5);
alarms.set(AlarmManager.RTC_WAKEUP, t.getTime(), rtcIntent);
// Fire intent in 30 mins if already awake.
alarms.set(AlarmManager.ELAPSED_REALTIME, 30*60*1000, elapsedIntent);
// Cancel the first alarm.
alarms.cancel(rtcIntent);
```