

# Android - Timer & AlarmManager

While both classes are designed to schedule tasks and repeating tasks one is meant to do it inside the activity or the service lifecycle and the other outside. The Java's Timer class meant to schedule tasks inside your components lifecycle. It's very simple and very basic. However the AlarmManager Android class is more complex and allows more flexibility. For example, you could use an alarm to initiate a long-running operation, such as starting a service once a day to download a weather forecast, this is done whether the user interact with your app or not. The user don't have to be in your app in order for the task to run, basically we schedule an alarm to the operating system and let it execute it's PendingIntent whenever the time arrives and even if the device itself is asleep. it can send broadcasts launch an activity or a service.

## Timer class

To use the timer class you first need to create a new Timer instance, after doing so you can use the schedule (for single task) or scheduleAtFixedRate(for repeating task) function and pass it three arguments the first one is an instance of TimerTask to be executed the second is the delay in millis before the first execution and the last is the interval in milliseconds between each execution. The TimerTask class basically contains the run method to be overridden and executed by the system when triggered. So a simple Timer can look like this:

```
Timer t = new Timer();
t.schedule(new TimerTask() {
    @Override
    public void run() {
        System.out.println("Run specific task after one second");
        t.cancel();
    }
}, 1000);
```

Or like this:

```
Timer t = new Timer();

t.scheduleAtFixedRate(new TimerTask() {

    @Override

    public void run() {

        System.out.println("Run this task every second!");

    }

}, 0, 1000);
```

**Note:** An important thing to remember is that you need to use the Handler class when performing UI task from within the TimerTask run method to send UI updates to the main thread.

## Canceling the timer

You can cancel the timer at any time by saving a reference to the timer and call cancel() on it

```
timer.cancel(); //Terminates this timer,discarding any currently scheduled tasks.
```

```
timer.purge(); // Removes all cancelled tasks from this timer's task queue.
```

## AlarmManager class

Alarms using AlarmManager have these characteristics:

- They let you fire Intents at set times and/or intervals (because the intent is fired by the system, it is wrapped with PendingIntent).
- You can use them in conjunction with broadcast receivers to start services and perform other operations.
- They operate outside of your application, so you can use them to trigger events or actions even when your app is not running, and even if the device itself is asleep.
- They help you to minimize your app's resource requirements. You can schedule operations without relying on timers or continuously running background services.

The first thing you need to do when setting an Alarm is to get the AlarmManager instance from the context by using:

```
AlarmManager manager = (AlarmManager) getSystemService(Context.ALARM_SERVICE);
```

After doing so you need to decide which type of alarm you need. Here is the list of types:

- [\*\*ELAPSED\\_REALTIME\*\*](#)—Fires the pending intent based on the amount of time since the device was booted, but doesn't wake up the device. The elapsed time includes any time during which the device was asleep.
- [\*\*ELAPSED\\_REALTIME\\_WAKEUP\*\*](#)—Wakes up the device and fires the pending intent after the specified length of time has elapsed since device boot.
- [\*\*RTC\*\*](#)—Fires the pending intent at the specified time but does not wake up the device.
- [\*\*RTC\\_WAKEUP\*\*](#)—Wakes up the device to fire the pending intent at the specified time.

For example:

```
manager.set(AlarmManager.ELAPSED_REALTIME_WAKEUP, 60*1000, pendingIntent);
```

This will set an alarm one minutes after **booting**. The alarm will fire the pendingIntent passed. If we want to trigger the alarm one minute from now we should do:

```
manager.set(AlarmManager.ELAPSED_REALTIME_WAKEUP, SystemClock.elapsedRealtime() + 60*1000, pendingIntent);
```

If we are using the RTC to trigger the alarm one minute from now we should use the :

```
alarmManager.set(AlarmManager.RTC_WAKEUP, System.currentTimeMillis() + 60*1000, pendingIntent);
```

In general the Elapsed clock is more reliable although rarely used by developers.

**Alarms can be exact or not**, if the alarm is inExact the alarm will not be delivered before this time, but may be deferred and delivered some time later. The OS will use this policy in order to "batch" alarms together across the entire system, minimizing the number of times the device needs to "wake up" and minimizing battery use. It is better practice not to schedule an exact alarm and let the OS manage the triggering of few events together when it's best for her and consume less battery power.

Prior to android 4.4 (API 19) the set method was exact but since then the set method is inExact and if we need an exact alarm we should use the **setExact(type,time,pendingIntent)**.

**Note:** if the time has already passed the alarm will be fired when posted.

## Schedule repeating alarms

Schedule alarms are more trickier, since API 19 the setRepeating() which also receives an interval for repeating alarms are treated as inExact and there is no way to make repeating alarm exact in the API while targeting to API 19(you can however target a lower SDK and it will work as Exact).

The only solution to this situation if we need the alarm to be exact is to use only the setExact and when receiving the alarm use setExact again to schedule the other one, meaning each alarm will schedule the next one.

Here is a simple code that triggers a broadcast receiver every one minutes that send an sms:

```
public class MainActivity extends AppCompatActivity {

    @Override

    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_main);

        AlarmManager alarm = (AlarmManager) getSystemService(Context.ALARM_SERVICE);

        Intent smsIntent = new Intent(MainActivity.this, AlarmReceiver.class);

        PendingIntent pending = PendingIntent.getBroadcast(MainActivity.this, 0, smsIntent, 0);

        if (Build.VERSION.SDK_INT > 18)

            alarm.setExact(AlarmManager.RTC_WAKEUP, System.currentTimeMillis() + 60 * 1000,

                pending);

        else

            alarm.set(AlarmManager.RTC_WAKEUP, System.currentTimeMillis() + 60 * 1000, pending);

    }

}
```

```

public class AlarmReceiver extends BroadcastReceiver { //The receiver which triggers

    @Override

    public void onReceive(Context context, Intent intent) {

        SmsManager sms = SmsManager.getDefault();

        sms.sendTextMessage("972509090900", null, "hello", null, null);

        AlarmManager alarm = (AlarmManager)context.getSystemService(Context.ALARM_SERVICE);

        PendingIntent pending = PendingIntent.getBroadcast(context, 0, intent, 0);

        if (Build.VERSION.SDK_INT > 18)

            alarm.setExact(AlarmManager.RTC_WAKEUP, System.currentTimeMillis() + 60 * 1000,
pending);

        else

            alarm.set(AlarmManager.RTC_WAKEUP, System.currentTimeMillis() + 60 * 1000, pending);

    }

}

```

## Canceling alarms

In order to cancel an alarm you should use the AlarmManager's cancel function and pass it the same PendingIntent you used to create the alarm (you can make him static to preserve this instance between sessions or create a new one with the same intent as before and same request code and use the 0 flag or UPDATE\_CURRENT)

For example

```

if (pending == null) {

    Intent smsIntent = new Intent(MainActivity.this, AlarmReceiver.class);

    pending = PendingIntent.getBroadcast(MainActivity.this, 0, smsIntent, 0);

}

alarmManager.cancel(pending);

```

**Reference:** Android Developer Guide - [Scheduling Repeating Alarms](#)