public class

AlarmManager

extends Object

java.lang.Object

↳android.app.AlarmManager

Class Overview

This class provides access to the system alarm services. These allow you to schedule your application to be run at some point in the future. When an alarm goes off, the `Intent` that had been registered for it is broadcast by the system, automatically starting the target application if it is not already running. Registered alarms are retained while the device is asleep (and can optionally wake the device up if they go off during that time), but will be cleared if it is turned off and rebooted.

The Alarm Manager holds a CPU wake lock as long as the alarm receiver's onReceive() method is executing. This guarantees that the phone will not sleep until you have finished handling the broadcast. Once onReceive() returns, the Alarm Manager releases this wake lock. This means that the phone will in some cases sleep as soon as your onReceive() method completes. If your alarm receiver called `Context.startService()`, it is possible that the phone will sleep before the requested service is launched. To prevent this, your BroadcastReceiver and Service will need to implement a separate wake lock policy to ensure that the phone continues running until the service becomes available.

Note: The Alarm Manager is intended for cases where you want to have your application code run at a specific time, even if your application is not currently running. For normal timing operations (ticks, timeouts, etc) it is easier and much more efficient to use `Handler`.

You do not instantiate this class directly; instead, retrieve it through `Context.getSystemService(Context.ALARM_SERVICE)`.

Summary

Constants		
int	ELAPSED_REALTIME	Alarm time in `SystemClock.elapsedRealtime()` (time since boot, includ
int	ELAPSED_REALTIME_WAKEUP	Alarm time in `SystemClock.elapsedRealtime()` (time since boot, includ will wake up the device when it goes off.

long	INTERVAL_DAY	
long	INTERVAL_FIFTEEN_MINUTES	Available inexact recurrence intervals recognized by `setInexactRepeatin` `long, PendingIntent)`
long	INTERVAL_HALF_DAY	
long	INTERVAL_HALF_HOUR	
long	INTERVAL_HOUR	
int	RTC	Alarm time in `System.currentTimeMillis()` (wall clock time in UTC).
int	RTC_WAKEUP	Alarm time in `System.currentTimeMillis()` (wall clock time in UTC), wh the device when it goes off.
Public Methods		
void	cancel(PendingIntent operation) Remove any alarms with a matching `Intent`.	
void	set(int type, long triggerAtTime, PendingIntent operation) Schedule an alarm.	
void	setInexactRepeating(int type, long triggerAtTime, long interval, PendingIntent operation) Schedule a repeating alarm that has inexact trigger time requirements; for example, an alarm that repeats eve necessarily at the top of every hour.	
void	setRepeating(int type, long triggerAtTime, long interval, PendingIntent operation) Schedule a repeating alarm.	
void	setTime(long millis) Set the system wall clock time.	
void	setTimeZone(String timeZone)	

	Set the system default time zone.

[Expand]
Inherited Methods

▸ From class java.lang.Object

Constants

public static final int ***ELAPSED_REALTIME***

Since: API Level 1

Alarm time in `SystemClock.elapsedRealtime()` (time since boot, including sleep). This alarm does not wake the device up; if it goes off while the device is asleep, it will not be delivered until the next time the device wakes up.
Constant Value: 3 (0x00000003)

public static final int ***ELAPSED_REALTIME_WAKEUP***

Since: API Level 1

Alarm time in `SystemClock.elapsedRealtime()` (time since boot, including sleep), which will wake up the device when it goes off.
Constant Value: 2 (0x00000002)

public static final long ***INTERVAL_DAY***

Since: API Level 3

Constant Value: 86400000 (0x0000000005265c00)

public static final long ***INTERVAL_FIFTEEN_MINUTES***

Since: API Level 3

Available inexact recurrence intervals recognized by `setInexactRepeating(int, long, long, PendingIntent)`
Constant Value: 900000 (0x00000000000dbba0)

public static final long ***INTERVAL_HALF_DAY***

Since: API Level 3

Constant Value: 43200000 (0x0000000002932e00)

public static final long ***INTERVAL_HALF_HOUR***

Since: API Level 3

Constant Value: 1800000 (0x00000000001b7740)

public static final long ***INTERVAL_HOUR***

Since: API Level 3

Constant Value: 3600000 (0x000000000036ee80)

public static final int ***RTC***

Since: API Level 1

Alarm time in `System.currentTimeMillis()` (wall clock time in UTC). This alarm does not wake the device up; if it goes off while the device is asleep, it will not be delivered until the next time the device wakes up.
Constant Value: 1 (0x00000001)

public static final int ***RTC_WAKEUP***

Since: API Level 1

Alarm time in `System.currentTimeMillis()` (wall clock time in UTC), which will wake up the device when it goes off.
Constant Value: 0 (0x00000000)

Public Methods

public void ***cancel*** *(PendingIntent operation)*

Since: API Level 1

Remove any alarms with a matching `Intent`. Any alarm, of any type, whose Intent matches this one (as defined by `filterEquals(Intent)`), will be canceled.

Parameters

operation	IntentSender which matches a previously added IntentSender.

See Also

- `set(int, long, PendingIntent)`

public void ***set*** *(int type, long triggerAtTime, PendingIntent operation)*

Since: API Level 1

Schedule an alarm. **Note: for timing operations (ticks, timeouts, etc) it is easier and much more efficient to use `Handler`.** If there is already an alarm scheduled for the same IntentSender, it will first be canceled.

If the time occurs in the past, the alarm will be triggered immediately. If there is already an alarm for this Intent scheduled (with the equality of two intents being defined by `filterEquals(Intent)`), then it will be removed and replaced by this one.

The alarm is an intent broadcast that goes to a broadcast receiver that you registered with `registerReceiver(BroadcastReceiver, IntentFilter)` or through the <receiver> tag in an AndroidManifest.xml file.

Alarm intents are delivered with a data extra of type int called `Intent.EXTRA_ALARM_COUNT` that indicates how many past alarm events have been accumulated into this intent broadcast. Recurring alarms that have gone undelivered because the phone was asleep may have a count greater than one when delivered.

Parameters

type	One of ELAPSED_REALTIME, ELAPSED_REALTIME_WAKEUP, RTC or RTC_WAKEUP.
triggerAtTime	Time the alarm should go off, using the appropriate clock (depending on the alarm type).
operation	Action to perform when the alarm goes off; typically comes from `IntentSender.getBroadcast()`.

See Also

- `Handler`
- `setRepeating(int, long, long, PendingIntent)`
- `cancel(PendingIntent)`
- `sendBroadcast(Intent)`
- `registerReceiver(BroadcastReceiver, IntentFilter)`
- `filterEquals(Intent)`
- `ELAPSED_REALTIME`
- `ELAPSED_REALTIME_WAKEUP`
- `RTC`
- `RTC_WAKEUP`

public void ***setInexactRepeating*** *(int type, long triggerAtTime, long interval, PendingIntent operation)*

Since: API Level 3

Schedule a repeating alarm that has inexact trigger time requirements; for example, an alarm that repeats every hour, but not necessarily at the top of every hour. These alarms are more power-efficient than the strict recurrences supplied by `setRepeating(int, long, long, PendingIntent)`, since the system can adjust alarms' phase to cause them to fire simultaneously, avoiding waking the device from sleep more than necessary.

Your alarm's first trigger will not be before the requested time, but it might not occur for almost a full interval after that time. In addition, while the overall period of the repeating alarm will be as requested, the time between any two successive firings of the alarm may vary. If your application demands very low jitter, use `setRepeating(int, long, long, PendingIntent)` instead.

Parameters

type	One of ELAPSED_REALTIME, ELAPSED_REALTIME_WAKEUP}, RTC or RTC_WAKEUP.
triggerAtTime	Time the alarm should first go off, using the appropriate clock (depending on the alarm type). This is inexact: the alarm will not fire before this time, but there may be a delay of almost an entire alarm interval before the first invocation of the alarm.
interval	Interval between subsequent repeats of the alarm. If this is one of INTERVAL_FIFTEEN_MINUTES, INTERVAL_HALF_HOUR, INTERVAL_HOUR, INTERVAL_HALF_DAY, or INTERVAL_DAY then the alarm will be phase-aligned with other alarms to reduce the number of wakeups. Otherwise, the alarm will be set as though the application had called `setRepeating(int, long, long, PendingIntent)`.
operation	Action to perform when the alarm goes off; typically comes from `IntentSender.getBroadcast()`.

See Also

- `Handler`
- `set(int, long, PendingIntent)`
- `cancel(PendingIntent)`
- `sendBroadcast(Intent)`
- `registerReceiver(BroadcastReceiver, IntentFilter)`
- `filterEquals(Intent)`
- `ELAPSED_REALTIME`
- `ELAPSED_REALTIME_WAKEUP`
- `RTC`
- `RTC_WAKEUP`
- `INTERVAL_FIFTEEN_MINUTES`
- `INTERVAL_HALF_HOUR`
- `INTERVAL_HOUR`
- `INTERVAL_HALF_DAY`
- `INTERVAL_DAY`

public void ***setRepeating*** *(int type, long triggerAtTime, long interval, PendingIntent operation)*

Since: API Level 1

Schedule a repeating alarm. **Note: for timing operations (ticks, timeouts, etc) it is easier and much more efficient to use `Handler`.** If there is already an alarm scheduled for the same IntentSender, it will first be canceled.

Like `set(int, long, PendingIntent)`, except you can also supply a rate at which the alarm will repeat. This alarm continues repeating until explicitly removed with `cancel(PendingIntent)`. If the time occurs in the past, the alarm will be triggered immediately, with an alarm count depending on how far in the past the trigger time is relative to the repeat interval.

If an alarm is delayed (by system sleep, for example, for non _WAKEUP alarm types), a skipped repeat will be delivered as soon as possible. After that, future alarms will be delivered according to the original schedule; they do not drift over time. For example, if you have set a recurring alarm for the top of every hour but the phone was asleep from 7:45 until 8:45, an alarm will be sent as soon as the phone awakens, then the next alarm will be sent at 9:00.

If your application wants to allow the delivery times to drift in order to guarantee that at least a certain time interval always elapses between alarms, then the approach to take is to use one-time alarms, scheduling the next one yourself when handling each alarm delivery.

Parameters

type	One of ELAPSED_REALTIME, ELAPSED_REALTIME_WAKEUP}, RTC or RTC_WAKEUP.
triggerAtTime	Time the alarm should first go off, using the appropriate clock (depending on the alarm type).
interval	Interval between subsequent repeats of the alarm.
operation	Action to perform when the alarm goes off; typically comes from `IntentSender.getBroadcast()`.

See Also

- `Handler`
- `set(int, long, PendingIntent)`
- `cancel(PendingIntent)`
- `sendBroadcast(Intent)`
- `registerReceiver(BroadcastReceiver, IntentFilter)`
- `filterEquals(Intent)`
- `ELAPSED_REALTIME`
- `ELAPSED_REALTIME_WAKEUP`
- `RTC`
- `RTC_WAKEUP`

public void ***setTime*** *(long millis)*

Since: API Level 8

Set the system wall clock time. Requires the permission android.permission.SET_TIME.

Parameters

millis	time in milliseconds since the Epoch

public void ***setTimeZone*** *(String timeZone)*

Since: API Level 1

Set the system default time zone. Requires the permission android.permission.SET_TIME_ZONE.

Parameters

timeZone in the format understood by `TimeZone`