public final class Summary: Methods | Inherited Methods | [Expand All]

Added in API level 1

SystemClock

extends Object

java.lang.Object
↳android.os.SystemClock

Class Overview

Core timekeeping facilities.

Three different clocks are available, and they should not be confused:

- `System.currentTimeMillis()` (/reference/java/lang/System.html#currentTimeMillis()) is the standard "wall" clock (time and date) expressing milliseconds since the epoch. The wall clock can be set by the user or the phone network (see `setCurrentTimeMillis(long)` (/reference/android/os/SystemClock.html#setCurrentTimeMillis(long))), so the time may jump backwards or forwards unpredictably. This clock should only be used when correspondence with real-world dates and times is important, such as in a calendar or alarm clock application. Interval or elapsed time measurements should use a different clock. If you are using System.currentTimeMillis(), consider listening to the `ACTION_TIME_TICK` (/reference/android/content/Intent.html#ACTION_TIME_TICK), `ACTION_TIME_CHANGED` (/reference/android/content/Intent.html#ACTION_TIME_CHANGED) and `ACTION_TIMEZONE_CHANGED` (/reference/android/content/Intent.html#ACTION_TIMEZONE_CHANGED) `Intent` (/reference/android/content/Intent.html) broadcasts to find out when the time changes.

- `uptimeMillis()` (/reference/android/os/SystemClock.html#uptimeMillis()) is counted in milliseconds since the system was booted. This clock stops when the system enters deep sleep (CPU off, display dark, device waiting for external input), but is not affected by clock scaling, idle, or other power saving mechanisms. This is the basis for most interval timing such as `Thread.sleep(millls)` (/reference/java/lang/Thread.html#sleep(long)), `Object.wait(millis)` (/reference/java/lang/Object.html#wait(long)), and `System.nanoTime()` (/reference/java/lang/System.html#nanoTime()). This clock is guaranteed to be monotonic, and is suitable for interval timing when the interval does not span device sleep. Most methods that accept a timestamp value currently expect the `uptimeMillis()` (/reference/android/os/SystemClock.html#uptimeMillis()) clock.

- `elapsedRealtime()` (/reference/android/os/SystemClock.html#elapsedRealtime()) and `elapsedRealtimeNanos()` (/reference/android/os/SystemClock.html#elapsedRealtimeNanos()) return the time since the system was booted, and include deep sleep. This clock is guaranteed to be monotonic, and continues to tick even when the CPU is in power saving modes, so is the recommend basis for general purpose interval timing.

There are several mechanisms for controlling the timing of events:

- Standard functions like `Thread.sleep(millis)` (/reference/java/lang/Thread.html#sleep(long)) and `Object.wait(millis)` (/reference/java/lang/Object.html#wait(long)) are always available. These functions use the `uptimeMillis()` (/reference/android/os/SystemClock.html#uptimeMillis()) clock; if the device enters sleep, the remainder of the time will be postponed until the device wakes up. These synchronous functions may be interrupted with `Thread.interrupt()` (/reference/java/lang/Thread.html#interrupt()), and you must handle `InterruptedException` (/reference/java/lang/InterruptedException.html).
- `SystemClock.sleep(millis)` (/reference/android/os/SystemClock.html#sleep(long)) is a utility function very similar to `Thread.sleep(millis)` (/reference/java/lang/Thread.html#sleep(long)), but it ignores `InterruptedException` (/reference/java/lang/InterruptedException.html). Use this function for delays if you do not use `Thread.interrupt()` (/reference/java/lang/Thread.html#interrupt()), as it will preserve the interrupted state of the thread.
- The `Handler` (/reference/android/os/Handler.html) class can schedule asynchronous callbacks at an absolute or relative time. Handler objects also use the `uptimeMillis()` (/reference/android/os/SystemClock.html#uptimeMillis()) clock, and require an `event loop` (/reference/android/os/Looper.html) (normally present in any GUI application).
- The `AlarmManager` (/reference/android/app/AlarmManager.html) can trigger one-time or recurring events which occur even when the device is in deep sleep or your application is not running. Events may be

scheduled with your choice of `currentTimeMillis()` (/reference/java/lang/System.html#currentTimeMillis()) (RTC) or `elapsedRealtime()` (/reference/android/os/SystemClock.html#elapsedRealtime()) (ELAPSED_REALTIME), and cause an `Intent` (/reference/android/content/Intent.html) broadcast when they occur.

Summary

Public Methods	
static long	currentThreadTimeMillis () Returns milliseconds running in the current thread.
static long	elapsedRealtime () Returns milliseconds since boot, including time spent in sleep.
static long	elapsedRealtimeNanos () Returns nanoseconds since boot, including time spent in sleep.
static boolean	setCurrentTimeMillis (long millis) Sets the current wall time, in milliseconds.
static void	sleep (long ms) Waits a given number of milliseconds (of uptimeMillis) before returning.
static long	uptimeMillis () Returns milliseconds since boot, not counting time spent in deep sleep.

Inherited Methods [Expand]

▸ From class java.lang.Object

Public Methods

public static long **currentThreadTimeMillis** () Added in API level 1

Returns milliseconds running in the current thread.

Returns

elapsed milliseconds in the thread

public static long **elapsedRealtime** () Added in API level 1

Returns milliseconds since boot, including time spent in sleep.

Returns

elapsed milliseconds since boot.

public static long **elapsedRealtimeNanos** ()

Added in API level 17

Returns nanoseconds since boot, including time spent in sleep.

Returns

elapsed nanoseconds since boot.

public static boolean **setCurrentTimeMillis** (long millis)

Added in API level 1

Sets the current wall time, in milliseconds. Requires the calling process to have appropriate permissions.

Returns

if the clock was successfully set to the specified time.

public static void **sleep** (long ms)

Added in API level 1

Waits a given number of milliseconds (of uptimeMillis) before returning. Similar to `sleep(long)` (/reference/java/lang/Thread.html#sleep(long)), but does not throw `InterruptedException` (/reference/java/lang/InterruptedException.html); `interrupt()` (/reference/java/lang/Thread.html#interrupt()) events are deferred until the next interruptible operation. Does not return until at least the specified number of milliseconds has elapsed.

Parameters

ms	to sleep before returning, in milliseconds of uptime.

public static long **uptimeMillis** ()

Added in API level 1

Returns milliseconds since boot, not counting time spent in deep sleep.

Returns

milliseconds of non-sleep uptime since boot.