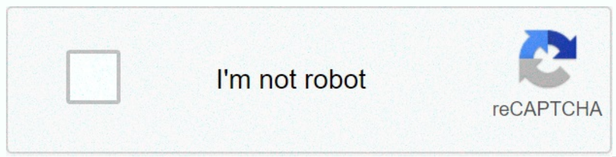

Continue

Android alarmmanager setinexactrepeating not working

Introduction Scheduling and repeating alarms are typically used as a local reminder to notify the user of certain events. Android provides AlarmManager for creating and scheduling alarms. The AlarmManager class provides access to the system's alarm service. This allows you to schedule the application to run at some point in the future. When the alarm goes off, the intent that was recorded for it is emitted by the system, automatically starting the target application if it is not already running. AlarmManager works outside the lifetime of the application. When an alarm is scheduled, it will be triggered even when the application is not running or in sleep mode. The scheduled alarm will execute unless it is explicitly stopped by calling cancel() or until the device is restarted. This means that you must rescheduly them explicitly after the device has finished booting. Alarms offer the following functions: Scheduling alarms for a fixed time or interval. Maintained by the operating system, not the application, so alarms are triggered even if the application is not running or the device is asleep. It can be used to trigger periodic tasks (such as hourly message updates), even if the application is not running. The application does not use resources (such as timers or background services) because the operating system manages scheduling. Alarms are not a best practice if you need a simple delay while your application is running, such as a short delay for a UI event. For short delays, it is easier and more efficient to use handler postAtTime() and postDelayed() methods. The alarm has three properties, as follows: Alarm type (see list below). Trigger time (if the time has already passed, the alarm is triggered immediately). Pending intents. A repeating alarm has the same three properties and an interval: alarm type (see list below). Trigger time (if the time has already passed, it triggers immediately). Interval. Pending intents. There are four types of alarms: RTC (Real Time Clock). Alarm times are referenced until UTC. This does not wake up the device. Runs the pending intent at the specified time. If the device is asleep, it will not be delivered until the next time the device wakes up. RTC_WAKEUP Alarm times are alluded to UTC time and resume the device to trigger if it is asleep. Starts the pending intent at a specific time, waking the device if it is asleep. ELAPSED_REALTIME. This is based on the time that has elapsed since the device was booted. This does not wake up the device. Interval alarms such as every 30 minutes are better. Starts the pending intent after a specified period of time after the device starts. If the device is asleep, it starts when the device is next to the ELAPSED_REALTIME_WAKEUP. This is based on the time that has elapsed since the device was booted. This wakes up your device if it's dormant. Starts the pending intent after a specified period of time after the device starts. Wakes up the device if it is asleep. RTC is most commonly used to set an alarm service on Android. Android. Choose the type of wake alarm, Android will wake your device from sleep, but it won't keep your device asleep for you. WakeLock from PowerManager should be obtained when performing background work from a wake-up event. Otherwise, Android will probably quickly put your device to sleep, which will stop what you can do. Thera are some methods that are used to schedule alarms. These methods are based on an accurate, inaccurate execution and repetition time or a single execution. set(int type, long triggerAtMillis, PendingIntent operation). Schedules an alarm, and if there is already an alarm by intent, the previous one will be canceled. Allows you to execute only once. setExact(int type, long triggerAtMillis, PendingIntent operation). It behaves the same as set(), but does not allow the operating system to adjust the time. The alarm will be fired in exactly the right time. setRepeating (type int, long triggerAtMillis, long intervalMillis, PendingIntent operation). Behaves the same as set(), but repeats for a given time interval. setInexactRepeating (type int, long triggerAtMillis, long intervalMillis, PendingIntent operation). Same as setRepeating(), but the execution time will not be accurate. setWindow(type int, long windowStartMillis, long windowLengthMillis, PendingIntent operation). Executes within a given time frame and works the same as set(). It can be used for strict ordering of execution for multiple alarms. After android API 19 (KitKat) all alarms are inaccurate, this is done to save battery life and disable multiple resumes. In this way, all alarms are associated with its near time and triggered at once. There are new APIs to support applications that require strict delivery guarantees; see setWindow() and setExact(). Applications whose targetSdkVersion is earlier than API 19 will still see the previous behavior, in which all alerts are delivered exactly when required. One-time alarm The following recipe will show you how to create alarms using AlarmManager. Setting an alarm requires a pending intent that Android sends when the alarm is triggered. This intent may point to any system component, such as BroadcastReceiver or service, that can be executed when an alarm is triggered. Therefore, we need to configure the receiving of the broadcast to intercept the intent of the alarm. Our user interface will consist of a simple button to set the alarm. To get started, open Android Studio and follow these steps: Add <receiver>the following items to an item at the same level as <application>existing <activity>item: <receiver android:name=. AlarmBroadcastReceiver> <intent-filter> <action android:name=me.proft.alarms.ACTION_ALARM></action> </intent-filter> </receiver> Open activity_main.xml and add the following button: xmlns:android= android:orientation=vertical android:padding=5dp android:layout_width=match_parent android:layout_height=match_parent> <Button android:id=@+id/btnAlarm android:layout_width=wrap_content android:id=@+id/btnAlarm android:layout_width=wrap_content></Button android:id=@+id/btnAlarm android:layout_width=wrap_content > </LinearLayout> </activity> </application> </receiver> </receiver> android:text=Set android alarm:layout_centerInParent=true android:onClick=setAlarm/> Create a new Java class named AlarmBroadcastReceiver using the following code: public class AlarmBroadcastReceiver extends BroadcastReceiver { public static final string ACTION_ALARM = en.proft.alarms.ACTION_ALARM; @Override public void onReceive(context context, intent) { if (ACTION_ALARM.equals(intent.getAction())) { Toast.makeText(context, ACTION_ALARM, Toast.LENGTH_SHORT).show(); } } } } Open ActivityMain.java and add a method to click: public void setAlarm(View view) { Intent IntentToFire = new Intent(getApplicationContext(), AlarmBroadcastReceiver.class); intentToFire.setAction(AlarmBroadcastReceiver.ACTION_ALARM); // pass something // Bundle bundle = new package(); // bundle.putString(TAG, FOO); // intentToFire.putExtra(BUNDLE, bundle); PendingIntent alarmIntent = PendingIntent.getBroadcast(getApplicationContext(), 0, intentToFire, 0); AlarmManager alarmManager = (AlarmManager)getApplicationContext(). getSystemService(Context.ALARM_SERVICE); Calendar c = Calendar.getInstance(); c.add(Calendar.MINUTE, 2); long afterTwoMinutes = c.getTimeInMillis(); alarmManager.set(AlarmManager.RTC, afterTwoMinutes, alarmIntent); } The alarm is created using this line of code: alarmManager.set(AlarmManager.ELAPSED_REALTIME, twoMinutes, alarmIntent); Here is the method signature: set(AlarmType, Time, PendingIntent); To set an alarm, we create a pending intent with our predefined alarm action: public static end ACTION_ALARM string = en.proft.alarms.ACTION_ALARM; This is any string and can be anything we want, but it must be unique, so we include our package name. We verify that this is an action in the callback of the broadcast receiver onReceive(). If you click set alarm and wait two minutes, you'll see a toast when the alarm starts. If you are too impatient to wait and click set alarm again before triggering the first alarm, you will not get two alarms. Instead, the operating system will replace the first alarm with a new alarm because both use the same pending intent. If you need multiple alarms, you need to create different pending intents, such as using different actions. If you want to cancel the alarm, call cancel(), passing the same pending intent that was used to create the alarm. If we continue our recipe, then what it will look like: alarmManager.cancel (alarmIntent); Repeating alarm If you want to create a repeating alarm, use the setRepeating() method. The signature is similar to the set() method, but with an interval. This is shown as follows: setRepeating(AlarmType, Time (in milliseconds), Interval, PendingIntent); For an interval, you can specify the interval time in milliseconds or use one of the predefined AlarmManager constants: INTERVAL_DAY INTERVAL_FIFTEEN_MINUTES INTERVAL_HALF_DAY INTERVAL_HOUR Let's write a very basic service it will simply display the current time as Toast every time it is launched. Public Class AlarmService extends service { @Override public int onStartCommand(Intent intent, int flags, int startId) { // display the current time Calendar now = Calendar.getInstance(); Formatter DataFormat = SimpleDateFormat.getTimeInstance(); Toast.makeText(this, formatter.format(now.getTime)), Toast.LENGTH_SHORT).show(); START_NOT_STICKY return flights; } @Override IBinder onBind(intent intent) { returns null; } } This service must be registered with AndroidManifest.xml <service> Tag. Otherwise, an AlarmManager that is external to the application will not be aware of how to trigger it. <application> ... <service android:enabled=true android:name=. AlarmService></service> </application> Below is the system to control our AlarmService. <?xml version=1.0 encoding=utf-8?>Here's an action to record/unregister an alarm. klasa publiczna AlarmActivity rozszerza AppCompatActivity implementuje View.OnClickListener { private PendingIntent alarmIntent; @Override publicznej void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState); setContentView(R.layout.main); // dołącz odbiornik do obu przycisków findViewById(R.id.start).setOnClickListener(this); findViewById(R.id.stop).setOnClickListener(this); // create the launch sender Intent launchIntent = new Intent(this, AlarmService.class); alarmIntent = PendingIntent.getService(this, 0, launchIntent, 0); } @Override public void onClick(View v) { AlarmManager manager = (AlarmManager)getSystemService(Context.ALARM_SERVICE); długi interwał = 60 * 1000; // 1 minuta switch(v.getId()) { case R.id.start: Toast.makeText(this, Scheduled, Toast.LENGTH_SHORT).show(); <LinearLayout xmlns:android= android:orientation=vertical android:layout_width=match_parent android:layout_height=match_parent> <Button android:id=@+id/start android:layout_width=match_parent android:layout_height=wrap_content android:text=Start Periodic Task></Button> <Button android :id=@+id/stop android:layout_width=match_parent android:layout_height=wrap_content android:text=Cancel Periodic Task></Button> </LinearLayout> Calendar c = Calendar.getInstance(); c.add(Calendar.SECOND, 10); long afterTenSeconds = c.getTimeInMillis(); manager.setRepeating(AlarmManager.RTC_WAKEUP, afterTenSeconds, interval, alarmIntent); break; R.id.stop case: Toast.makeText(this, Canceled, Toast.LENGTH_SHORT).show(); manager.cancel(alarmIntent); break; default: break; } } } From Android 5.1 (API version 22) there is a minimum period of 1 minute for repeated alarms. If you want to do the work in one minute, just set the alarm directly, and then set the next one with the handler of this alarm, etc. If you want to get the job done within 5 seconds (for example), publish it to the Handler instead of using an alarmmanager. An example action shows two buttons: one to start firing regular alarms and the other to cancel them. Trigger operation </service> </service> refers to PendingIntent, which will be used to both set and cancel alarms. We create an intent to refer to the service directly, and then wrap that intent inside the PendingIntent obtained from getService(). The alarm in the example is recorded to trigger 1 minute after pressing the button, and then every 1 minute after that. Every 1 minute, Toast will appear on the screen with the current time value, even if the application is no longer running or in front of the user. When the user views the activity and presses the Stop button, any pending alarms matching our Pending Intent will be immediately canceled and stop the flow of toast. Precision alarm What if we want to schedule an alarm to occur within a certain time? Perhaps exactly at 14:00? Setting the AlarmManager with slightly different parameters can achieve this. download the calendar and set the calendar start time to 14:00:00Time = Calendar.getInstance(); startTime.set(Calendar.HOUR_OF_DAY, 14); startTime.set(Calendar.MINUTE, 0); download calendar at current time Calendar now = Calendar.getInstance(); if (now.before(startTime)) { // is not yet 14:00, start today time = startTime.getTimeInMillis(); } else { // start 14:00 tomorrow startTime.add(Calendar.DATE, 1); time = startTime.getTimeInMillis(); } // set the alarm if (Build.VERSION.SDK_INT < Build.VERSION_CODES. KITKAT) { alarmManager.set(AlarmManager.RTC, time, alarmIntent); } else { alarmManager.setExact(AlarmManager.RTC, time, alarmIntent); } else { alarmManager.setExact(AlarmManager.RTC, time, alarmIntent); } This example uses an alarm referenced by real time. It is determined whether the next occurrence at 14:00 will be today or tomorrow, and this value is returned as the alarm trigger time. Starting with Android 4.4, AlarmManager by default all alarms are inaccurate, which means there is a small window in which they will trigger. With this new behavior, the setExact() API method has been added to allow developers to declare that the following alarm cannot fit in an inaccurate window. Before 4.4, it was enough just to call set() with the appropriate start time. Set alarms in the Android API level 23 and higher If our app is designed for API level below 19 (KitKat), scheduled alerts will work at the exact time of the alarm. For applications designed for KitKat or later, the schedule is considered inaccurate, and the system can re-order or group alarms to minimize resuming and save battery. After API level 23, the Android development team went a little further, and Doze mode was introduced on Android to reduce battery consumption when the device is disconnected from the power supply, stationary and not used by the user for a long period of time. The Doze system will try to reduce the frequency of device resuming by de-snooze background tasks, networking, synchronizations, and our valuable alarm until your device exits Doze mode, or a recurring maintenance window is launched to perform pending tasks, some alarms, or network synchronization. After maintenance is complete terminates, the device enters Doze mode again if it has not been used in the meantime: Doze mode can affect the application and defer alarms until a maintenance window appears unless you use the setAndAllowWhileIdle() and setExactAndAllowWhileIdle() methods to allow idle alarms to be executed. What's more, the number of Doze maintenance window starts will be less frequent in case of prolonged inactivity, so the impact of this new mechanism on our schedule will increase, thus causing more unpredictable jitters during the alarm. In doze mode, applications also cannot access the network, WakeLocks are ignored and Wi-Fi scans are not performed. If we need precise planning and you are directing marshmallow or later, we will use the new setExactAndAllowWhileIdle() method introduced at API level 23: am.setExactAndAllowWhileIdle(AlarmManager.RTC, time, pending); Android has protection that prevents abuse for accurate alarms that fire too often. AlarmManager will build the device and send only one alarm per minute, and in low-notch mode it can be as low as one every 15 minutes. If your application is intended for versions between KitKat (API level 19) and Marshmallow (API level 23), the setExact method is sufficient for timing precision: am.setExact(AlarmManager.RTC, pending time); But we need to see if the methods exist before we try to call it; otherwise, our application will crash when run within earlier API levels. Allows you to sketch our new exact alarm code: if (Build.VERSION.SDK_INT > = Build.VERSION_CODES. M) { // Wakes up the device in Doze am.setExactAndAllowWhileIdle(AlarmManager.RTC, time, pending); } else if (Build.VERSION.SDK_INT >= Build.VERSION_CODES. LOLLIPOP) { // Wakes up the device in idle mode am.setExact(AlarmManager.RTC_WAKEUP, time, wait); } still { // Old APIs am.set(AlarmManager.RTC, time, in progress); } other {// Old APIs am.set(AlarmManager.RTC, time, waiting); } other {// Old APIs am.set(AlarmManager.RTC, time, waiting); } This will deliver our alarm at exactly the specified time on all platforms. Don't forget that you should use accurate scheduling only when you really need it, for example, to deliver alerts to the user at a specific time. In most other cases, allowing the system to slightly adjust our schedule to maintain battery life is usually acceptable. Android Marshmallow API Level 23 is also equipped with the setAndAllowWhileIdle function, which allows us to create an alarm to sound in Doze mode, but with less accuracy compared to setExactAndAllowWhileIdle(). The system will try to batch this type of alarm throughout the system, minimizing the number of times the device wakes up, thereby reducing the energy consumption of the system. Here is the code to create an alarm that triggers, even in Doze, 10 hours from now: long delay = TimeUnit.HOURS.toMillis(10L); long time = System.currentTimeMillis() + delay; if (Build.VERSION.SDK_INT > = Build.VERSION_CODES. M) { am.setAndAllowWhileIdle(AlarmManager.RTC, time, in progress); } To test Behavior in doze mode, the Android SDK team has added some new commands to the dumpsys tool to manually change the power state of the device from the command line. It is also important to note that Doze mode requires the device to be disconnected from the charger. To force the device to a state where it is disconnected from the charger, we should run the following command on the command line with access to SDK tools: # Emulate the charger disconnect adb shell dumpsys battery disconnected # Emulation of the charger plug in adb shell dumpsys battery set ac 1 Then, to go into idle mode, we should turn off the screen and run the following commands: // Enable doze mode, step required on Android Emulator adb shell dumpsys deviceidle enable // To go directly into IDLE mode adb shell dumpsys deviceidle force-idle After placing the device in idle mode, we can enable the maintenance window by running the following command: // Goes with IDLE -> IDLE_MAINTENANCE state adb shell dumpsys deviceidle step If we run the same step again the device returns to idle; however, if we want to return to the active state, we should run the next command: // Goes with IDLE,IDLE_MAINTENANCE -> ACTIVE state adb shell dumpsys deviceidle disable Thanks to this handy command we are able to check if the alarm sounds even in deep idle states. Restarting the Alarm Service when the device restarts As mentioned earlier, when the alarm service starts, it is performed until it is explicitly stopped or until the device is restarted. This means that if the device is restarted, the alarm will stop. To avoid this situation, restart the emergency service as soon as the device starts. The following snippet will help you start the alarm service when you restart your device. Public class DeviceBootReceiver extends BroadcastReceiver { @Override public void onReceive(context context, intent) { if (intent.getAction().equals(android.intent.action.BOOT_COMPLETED)) { Intent alarmIntent = new Intent(context, AlarmReceiver.class); PendingIntent pendingIntent = PendingIntent.getBroadcast(context, 0, alarmIntent, 0); AlarmManager manager = (AlarmManager) context. getSystemService(Context.ALARM_SERVICE); interval int = 8000; manager.setInexactRepeating(AlarmManager.RTC_WAKEUP, System.currentTimeMillis(), interval, pendingIntent); Toast.makeText(context, Alarm set, Toast.LENGTH_SHORT).show(); } } } To trigger an alarm after the device restarts, you must register the above-declared DeviceBootReciever class in the application manifest. This also requires android.permission.RECEIVE_BOOT_COMPLETED. <application> <uses-permission android:name=android.permission.RECEIVE_BOOT_COMPLETED></uses-permission> ... How do I know if an alarm has already been scheduled or not? android:name=. DeviceBootReceiver android:enabled=false> <intent-filter> <action android:name=android.intent.action.BOOT_COMPLETED></action> </intent-filter> </receiver> </application> Utwórz odpowiednik PendingIntent, który został użyty z setRepeating() z z Flag. Intent intent = new intent(context, WidgetUpdateReceiver.class); PendingIntent pendingIntent = PendingIntent.getBroadcast(context, 0, intent, PendingIntent.FLAG_NO_CREATE); if (pendingIntent != null) { Log.d(TAG, Alarm is already active); } When using a PendingIntent.FLAG_NO_CREATE, if the described PendingIntent does not already exist, it simply returns null. How to obtain wakelock WAKE_LOCK permission is required because a wake lock is used when processing in the onReceive() method present in the BroadcastReceiver class. <uses-permission android:name='android.permission.WAKE_LOCK'></uses-permission> <application> ... <receiver android:name='. AlarmBroadcastReceiver'></receiver> </application> WakeLock example for BroadcastReceiver. Public class AlarmBroadcastReceiver extends BroadcastReceiver { @Override public void onReceive(context context, intent) { PowerManager pm = (PowerManager) context.getSystemService(Context.POWER_SERVICE); PowerManager.WakeLock wl = pm.newWakeLock(PowerManager.PARTIAL_WAKE_LOCK, TAG); acquire(); you can perform processing here. ... // release the release() lock; } } Replace all alarms on your device You may know that the alarm has been set and when they will alarm and interval. Also how many times this alarm was triggered. # from terminal adb device # connect to emulator adb connect emulator-5554 # alarm list adb -s emulator-5554 dump shellsys alarm Scheduler scheduler is a great candidate for scheduling if the application needs to perform a local event at the exact time or inaccurate interval. Alarm clock or reminder apps are great examples for using AlarmManager. However, documentation discourages using AlarmManager to schedule network-related tasks. For these tasks, you can use JobScheduler, which is an API for scheduling different types of tasks against the structure that will be performed in your own application process. Process.