Alarmmanager in android studio

By downloading the video, you agree to YouTube Privacy Policy. Learn more Download video Always unblock YouTube package com.example.application.alarmmanagerproject; import Android.app.AlarmManager; import Android.app.PendingIntent; import Android.app.TimePickerDialog; import android.content.Context; import android.content.Intent; import android.support.v4.app.DialogFragment; import android.support.v7.app.AppCompatActivity; import android.os.Bundle; import android.view.View; import android.widget.Button; import android.widget.TextView; import android.widget.TimePicker; import java.text.DateFormat; import java.util.Calendar; public class MainActivity extends AppCompatActivity implements TimePickerDialog.OnTimeSetListener { private TextView mTextView; @Override protected void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState); setContentView(R.layout.activity_main); mTextView find =ViewById(R.id.textView); Button ButtonTimePicker = findViewById(R.id.button_timepicker); buttonTimePicker.setOnClickListener(new View.OnClickListener() { @Override public void onClick(View v) { DialogFragment timePicker = new TimePickerFragment(); timePicker.show(getSupportFragmentManager(), time picker); } }); Button buttonCancelAlarm = findViewById(R.id.button_cancel); buttonCancelAlarm.setOnClickListener(new View.OnClickListener() { @Override public void onClick(View v) { cancelAlarm(); } }); } @Override public void optionTimeSet(TimePicker view, int hourOfDay, int minute) { Calendar c = Calendar.getInstance(); c.set(Calendar.HOUR_OF_DAY, hourOfDay); c.set(Calendar.MINUTE, minute); c.set(Calendar.SECOND, 0); updateTimeText(c); startAlarm(c); } private void updateTimeText(Calendar c) { String timeText = Alarm set for : ; timeText += DateFormat.getTimeInstance(DateFormat.SHORT).format(c.getTime()); mTextView.setText(timeText); } private void StartAlarm(Calendar c) { AlarmManager alarmManager = (AlarmManager) getSystemService(Context.ALARM_SERVICE); Intent intent = new Intent(this, AlertReceiver.class); PendingIntent pendingIntent = PendingIntent.getBroadcast(this, 1, opt, 0); if (c.before(Calendar.getInstance())) { c.add(Calendar.DATE, 1); } alarmManager.setExact(AlarmManager.RTC_WAKEUP, c.getTimeInMillis(), pendingIntent); } private void cancelAlarm() { AlarmManager alarmManager = (AlarmManager) getSystemService(Context.ALARM_SERVICE); Intent intent = new Intent(this, AlertReceiver.class); PendingIntent pendingIntent = PendingIntent.getBroadcast(this, 1, opt, 0); alarmManager.cancel(pendingIntent); mTextView.setText(Alarm cancelled); } This class provides access to the system alarm services. This allows you to schedule your application to be run at some point in the future. When an alarm goes off, the intention that for it was broadcast by the system, which automatically starts the target application if it is not already running. not. alarms are retained while the device is asleep (and can optionally wake up the device if they go off during that time), but will be cleaned if turned off and recharged. The Alarm Manager has a CPU wake lock as long as the alarm executes receiver's onReceive() method. This guarantees that the phone won't sleep until you've finished handling the broadcast. Once onReceive() returns, the Alarm Manager releases this wake lock. This means that the phone will sleep in some cases once your unreceive() method completes. If your alarm receiver is named Context.startService(), it is possible that the phone will sleep before launching the requested service. To prevent this, your BroadcastReceiver and Service will need to implement a separate wake-lock policy to ensure the phone keeps running until the service becomes available. Note: The Alarm Manager is intended for instances where you want to run your application code at a specific time, even if your application is not currently running. For normal timing operations (ticks, timeouts, etc.) it's easier and much more efficient to use Handler. Note: Get started with API 19 (Build.VERSION_CODES. KITKAT) alarm delivery is inaccunam: the Industry will shift alarms to reduce wake and battery use. There are new APIs to support applications that require strict delivery guarantees; see setWindow(int, long, long, Android.app.PendingIntent) and setExact(int, long, Android.app.PendingIntent). Applications whose targetSdkVersion is earlier than API 19 will continue to see the previous behavior in which all alarms are rendered exactly when requested. class AlarmManager.AlarmClockInfo An immutable description of a scheduled alarm clock event. interface AlarmManager.OnAlarmListener Direct notification alarms: the requester must constantly run from the time the alarm is set to the time it is delivered, or delivery will fail. String ACTION_NEXT_ALARM_CLOCK_CHANGED Action: Sent after changing the value returned by getNextAlarmClock(). int ELAPSED_REALTIME Alarm time in SystemClock.elapsedRealtime() (time since boat, including sleep). int ELAPSED_REALTIME_WAKEUP Alarm time in SystemClock.elapsedRealtime() (time since boat, including sleep), which will wake up the device when it goes down. long INTERVAL_DAY Available inactive recurrence interval recognized by setInexactRepeating(int, long, Android.app.PendingIntent) when running on Android before API 19. long INTERVAL_FIFTEEN_MINUTES Available inactive recurrence interval recognized by setInexactRepeating(int, long, long, Android.app.PendingIntent) when running on Android before API 19. long INTERVAL_HALF_DAY Available inactive recurrence interval recognized by setInexactRepeating (int, long, long, Android.app.PendingIntent) run on Android before API 19. long INTERVAL_HALF_HOUR Available inaccoon inacconveni interval recognized by setInexactRepeating(int, long, long, android.app.PendingIntent) when running on Android before API 19. long INTERVAL_HOUR Available inactive recurrence interval recognized by setInexactRepeating(int, long, long, Android.app.PendingIntent) when running on Android before API 19. 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), which will wake the device up when it goes off. void Cancel(AlarmManager.OnAlarmListener listener) Remove any alarm scheduled to be delivered to the given OnAlarmListener. AlarmManager.AlarmClockInfo getNextAlarmClock() Get information about the following alarm clock currently scheduled. void set(int type, long triggerAtMillis, PendingIntent operation) Schedule an alarm. void set(int type, long triggerAtMillis, String tag, AlarmManager.OnAlarmListener listener, Handler targetHandler) Direct callback version of set(int, long, Android.app.PendingIntent). void setAlarmClock(AlarmManager.AlarmClockInfo information, PendingIntent operation) Schedule an alarm that represents an alarm clock, which will be used to notify the user when it goes off. long triggerAtMillis, PendingIntent operation) Like set(int, Long, Android.app.PendingIntent), but this alarm will be allowed to execute even when the system is in low-power diaper (a.k.a. void setExact(int type, long triggerAtMillis, PendingIntent operation) Schedule an alarm to be rendered exactly on the state long triggerAtlisMillis, String tag, AlarmManager.OnAlarmListener listener, Handler targetHandler) Direct Callback Version of setExact(int, long, android.app.PendingIntent)., long triggerAtMillis, PendingIntent operation) Like setExact(int, long, android.app.PendingIntent), but this alarm will be allowed to execute even when the system is in low-power idle modes. , long triggerAtMillis, long intervalMillis, PendingIntent operation) Schedule a repeat alarm that has inactive trigger time requirements; for example, an alarm that repeats every hour, but not necessarily at the top of each hour. void setRepeating(int type, long triggerAtMillis, long intervalMillis, PendingIntent operation) Schedule a repeat alarm. void setTime(long millis) Set the system wall clock time. void setTimeZone(String timeZone) Set the system's persistent default time zone. void setWindow(int type, long windowStartMillis, long windowLengthMillis, PendingIntent operation) Schedule an alarm to be delivered within a given window of time. void setWindow(int type, long windowStartMillis, long windowLengthMillis, String tag, AlarmManager.OnAlarmListener listener, Handler Direct callback version of setWindow(int, long, long, Android.app.PendingIntent). From class java.lang.Object Object clone() Create and return a copy of this object. boolean equal (Object obj) Indicates whether another object is equal to this one. void finalizes() Called by the garbage collector on an object when garbage collection determines that there are no more references to the object. final Class<?> getClass() Returns the runtime class of this Object. int hashCode() Returns a hash code value for the object. final void set() Wake up to a single thread awaiting this object's monitor. String toString() Returns a string representation of the object. final void await(long timeout, int nanos) Causes the current thread to wait until another thread invokes the notification() method or the notifyingAll() method for this object, or another thread interrupts the current thread, or a certain amount of real time has elapsed. final void await(long timeout) Causes the current thread to wait until either another thread invokes the notify() method or the notifyingAll() method for this object, or a specified amount of time has elapsed. final void await() Causes the current thread to wait until another thread invokes the notify() method or the notifyIngAll() method for this object. public static final String ACTION_NEXT_ALARM_CLOCK_CHANGED Action: Sent after changing the value returned by getNextAlarmClock(). This is a protected intention that can only be sent through the system. It is sent only to registered recipients. Constant value: android.app.action.NEXT_ALARM_CLOCK_CHANGED public static final int ELAPSED_REALTIME Alarm time in SystemClock.elapsedRealtime() (time since boat, including sleep). This alarm does not wake up the device; if it goes off while the device is asleep, it won't be delivered until the next time the device wakes up. Constant value: 3 (0x0000003) public static final int ELAPSED_REALTIME_WAKEUP Alarm time in SystemClock.elapsedRealtime() (time since boating, including sleep), which the device will wake up when it goes down. Constant value: 2 (0x0000002) public static final in RTC Alarm time in System #currentTimeMillis (wall in UTC). This alarm does not wake up the device; if it goes off while the device is asleep, it won't be delivered until the next time the device wakes up. Constant value: 1 (0x0000001) public static final int RTC_WAKEUP Alarm time in System #currentTimeMillis (wall clock time in UTC), which will wake the device up when it goes down. Constant value: 0 (0x0000000) public void set(int type, long triggerAtMillis, PendingIntent operation) Schedule an alarm. for timing operations (ticks, timeouts, etc.) it is easier and much more efficient to use Handler. If there is already an alarm for the same that previous alarm will only be cancelled. If the declared trigger time is in the past, the alarm will be caused immediately. If there is already an alarm scheduled for this Intention (with the equality of two intentions defined by Intent #filterEquals), it will be removed and replaced by this one. The alarm is an Intention broadcast that goes to a broadcast recipient that you registered with Context.registerReceiver(BroadcastReceiver, IntentFilter) or by <receiver > the tag in an AndroidManifest.xml file. Alarm intents are delivered with a data extra type of int called Intent #EXTRA_ALARM_COUNT indicating how many previous alarm events have been accumulated in this intent. Recurring alarms that went unleaded because the phone was asleep may have a score bigger than one when delivered. Note: Starting in API 19, the trigger time passed to this method is treated as inaccurate: the alarm will not be delivered before this time, but can be postponed and delivered some time later. The OS will use this policy to batch alarms across the entire system, reducing the number of times the device needs to wake up and reduce battery usage. In general, alarms scheduled in the near future will not be postponed as long as alarms are scheduled far into the future. With the new batch policy, shipping order guarantees aren't as strong as they were before. If the application sets multiple alarms, it's possible that these alarms' actual delivery order doesn't match the order of their requested delivery times. If your application has strong order requirements, there are other APIs that you can use to get the necessary behavior; see setWindow(int, long, long, Android.app.PendingIntent) and setExact(int, long, Android.app.PendingIntent). Applications whose targetSdkVersion is ahead of API 19 will continue to get the previous alarm behavior: all of their scheduled alarms will be treated as precisely. public void set(int type, long triggerAtMillis, String tag, AlarmManager.OnAlarmListener listener, Handler targetHandler) Direct callback version of set(int, long, Android.app.PendingIntent). Rather than providing a PendingIntent to be sent when the alarm time is reached, this variant provides an OnAlarmListener instance that will be invoked at that time. The OnAlarmListener's OnAlarmListener#onAlarm() method will be invoked via the specified target handler, or on the application's main cluster if null is passed as the targetHandler parameter. Parameters type int: type alarm. Value is RTC_WAKEUP, RTC, ELAPSED_REALTIME_WAKEUP or ELAPSED_REALTIME triggerAtMillis long: time in milliseconds that the alarm should go off, using the appropriate clock of the alarm type). tag String: string describes the alarm, used for logging in and battery-using attribution listener AlarmManager.OnAlarmListener: OnAlarmListener case </receiver> </receiver > OnAlarmListener#onAlarm() method will be called when the alarm time is reached. A given OnAlarmListener instance can only be the target of a single pending alarm, just as a given PendingIntent can only be used with one alarm at a time. targetHandler Handler: Handler on which to perform the listener's onAlarm() callback, or zero to perform that callback on the main looper. public void setAlarmClock(AlarmManager.AlarmClockInfo information, PendingIntent operation) Schedule an alarm that represents an alarm watch, which will be used to notify the user when it goes down. The expectation is that when this alarm triggers, the application will further wake up the device to tell the user about the alarm - turn on the screen, play a sound, vibrating, etc. As such, the system will usually also use the information provided here to tell the user of this upcoming alarm if appropriate. Due to the nature of this kind of alarm, similar to setExactAndAllowWhileIdle (int, long, PendingIntent), these alarms will be allowed to activate even if the system is in low-power idle (a.k.a. doze) mode. The system can also do some prep work when it sees such an alarm coming, to avoid the amount of background work that can happen if it causes the device to wake up fully - it's to avoid situations like a large number of devices with an alarm that at the same time in the morning, all waking up at the time and suddenly swamping the network with pending background work. As such, these types of alarms can be extremely expensive on battery use and should only be used for their intended purpose. This method is like setExact(int, long, Android.app.PendingIntent), but implies RTC_WAKEUP. See also: set(int, long, PendingIntent)setRepeating(int, long, long, PendingIntent)setWindow(int, long, long, PendingIntent)setExact(int, long, PendingIntent)cancel(AlarmManager.OnAlarmListener)getNextAlarmClock()Context.sendBroadcast(Intent)Context.RegisterReceiver(BroadcastReceiver, IntentFilter)Intent.filterEquals(Intent) public void setAndAllowWhileIdle(int type, long triggerAtMillis, PendingIntent operation) As set (int, long, android.app.PendingIntent), but this alarm will be allowed to execute even when the system is in low-power idle (a.k.a. doze) modes. This type of alarm should only be used for situations where it is actually required that the alarm go off while in idle - would be a reasonable example for a calendar notification that should make a sound so that the user is aware of it. When the alarm is dispatched, the app will also be added to the system's temporary whitelist for about 10 seconds to allow that application to acquire further wacky locks in which she works 10.2.2. These alarms can have a significant impact on the power use of the device when idle (and thus cause significant battery blame to the app scheduling them), so they should be used with care. 2019 2019 Abuse, there are restrictions on how frequently these alarms will go down for a particular application. Under normal system operation, it will not dispatch these alarms more than about every minute (at which point each such pending alarm is dispatched); when in low-power diaper modes this duration can be considerably longer, like 15 minutes. Unlike other alarms, the system is free to reschedule this type of alarm to get out of operation with any other alarms, even that of the same program. This will clearly happen when the device is idle (since this alarm can go off while idle, when any other alarms from the app will be kept until later), but can also happen even when it is not idle. Regardless of the app's target SDK version, this call always allows the batch of the alarm. public void setExact(int type, long triggerAtMillis, PendingIntent operation) Schedule an alarm to be rendered exactly at the set time. This method is as set (int, long, android.app.PendingIntent), but does not allow the OS to adjust delivery time. The alarm will be delivered to the requested trigger time as almost as possible. Note: only alarms for which there is a strong demand for precise delivery (such as an alarm bell ring at the requested time) should be scheduled as precisely. Applications are strongly discouraged from using precise alarms unnecessarily as they reduce the OS's ability to reduce battery usage. public void setExact(int type, long triggerAtMillis, String tag, AlarmManager.OnAlarmListener listener, Handler targetHandler) Direct callback version of setExact(int, long, android.app.PendingIntent). Rather than providing a PendingIntent to be sent when the alarm time is reached, this variant provides an OnAlarmListener instance that will be invoked at that time. The OnAlarmListener's OnAlarmListener#onAlarm() method will be invoked via the specified target handler, or on the application's main cluster if null is passed as the targetHandler parameter. public void setExactAndAllowWhileIdle(int type, long triggerAtMillis, PendingIntent operation) Such as setExact(int, long, android.app.PendingIntent), but this alarm will be allowed to execute even when the system is in low-power idle modes. If you don't need exact scheduling of the alarm, but still while idle, consider using setAndAllowWhileIdle (int, long, PendingIntent). This type of alarm should only be used for situations where it is actually required that the alarm go off while in idle - would be a reasonable example for a calendar notification that should make a sound so that the user is aware of it. When the alarm is dispatched, the app will also be added to the the temporary whitelist is added to allow that application to obtain further wacky locks in which his work can be completed. These alarms can cause a significant impact on the power use of the device when idle (and therefore caused battery blame to the app scheduling them), so they should be used with care. To reduce abuse, there are restrictions on how frequently these alarms will go down for a particular application. Under normal system operation, it will not dispatch these alarms more than about every minute (at which point each such pending alarm is dispatched); when in low-power diaper modes this duration can be considerably longer, like 15 minutes. Unlike other alarms, the system is free to reschedule this type of alarm to get out of operation with any other alarms, even that of the same program. This will clearly happen when the device is idle (since this alarm can go off while idle, when any other alarms from the app will be kept until later), but can also happen even when it is not idle. Note that the Industry will allow itself more flexibility to schedule these alarms than regular exact alarms, since the application has chosen in this behavior. When the device is idle, it can take even more freedoms with scheduling to optimize for battery life. public void setInexactRepeating(int type, long triggerAtMillis, long intervalMillis, PendingIntent operation) Schedule a repeat alarm that has inactive trigger time requirements; for example, an alarm that repeats every hour, but not necessarily at the top of each hour. These alarms are more power-efficient than the strict repetitions traditionally provided by setRepeating (int, lung, long, PendingIntent), as the system can adjust alarms' delivery times to cause them to fire simultaneously, and avoid waking the device more than necessary. Your alarm's first trigger won't be ahead of the requested time, but it might not occur for almost a full interval after that time. In addition, while the overall period of the recurring alarm will be as requested, the time between any two consecutive firings of the alarm may vary. If your application requires very low jitter, use one-shot alarms with an appropriate window instead; see setWindow(int, long, long, Android.app.PendingIntent) and setExact(int, long, Android.app.PendingIntent). Due to API 19, all repeated alarms are inacqueone. Because this method has been available since API 3, your application can call it safe and be assured that it will get similar behavior on both current and older versions of Android. Parameters type int: type alarm. Value is RTC_WAKEUP, RTC, ELAPSED_REALTIME_WAKEUP or ELAPSED_REALTIME triggerAtMillis long: time in milliseconds that the alarm must first go off, using the appropriate clock (depending on the alarm type). It's inaccurate: the alarm won't fire ahead of this time, but there could be a delay of almost an entire alarm interval before the first invocation of the alarm. Long: interval in milliseconds between subsequent repetitions of the alarm. Before API 19, if it is one of the INTERVAL_FIFTEEN_MINUTES, INTERVAL_FIFTEEN_MINUTES, INTERVAL_HOUR, INTERVAL_HALF_DAY or INTERVAL_DAY the alarm will be phase-aligned with other alarms to reduce the number of wakes. Otherwise, the alarm will be set as if the application has called setRepeating (int, long, long, PendingIntent). Starting from API 19, all recurring alarms will be inaccurate and subject to bundles with other alarms regardless of their stated repeated interval. operation PendingIntent: Action to perform when the alarm goes off; typically comes from PendingIntent #getBroadcast. public void setRepeating(int type, long triggerAtMillis, long intervalMillis, PendingIntent operation) Schedule a repeat 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 be cancelled first. As set (int, long, PendingIntent), except that you can also provide a period at which the alarm will automatically replicate. This alarm still repeats until explicitly removed with cancel (AlarmManager.OnAlarmListener). If the declared trigger is time in the past, the alarm will be caused immediately, with an alarm count depending on how far in the past the trigger time is relative to the repeated interval. If an alarm is delayed (by system sleep, for example, for non-_WAKEUP alarm types), a skipped recurrence will be delivered as soon as possible. Thereafter, future alarms will be delivered according to the original schedule; they don't float over time. For example, if you set a recurring alarm for the top of each hour, but the phone was asleep from 7:45 a.m. to 8:45 p.m., an alarm will be sent once the phone awakens, then the next alarm will be sent at 9 p.m. If your application wants to allow delivery times to drift away to guarantee that at least a certain time interval always runs between alarms, then the approach to take is to use one-off alarms, which schedule the next one itself when handling each alarm delivery. Note: from api 19, all recurring alarms are inaccurate. If your application requires exact delivery times, it should use one-time exact alarms, which reschedule each time as described above. Legacy applications whose targetSdkVersion is earlier than API 19 will continue to have all their alarms, including repeat alarms, treated as precise. Parameters type int: type alarm. Value is RTC_WAKEUP, RTC, ELAPSED_REALTIME_WAKEUP or ELAPSED_REALTIME triggerAtMillis long: time in milliseconds that the alarm must first go off, using the appropriate clock (depending on the alarm type). intervalMillis long: interval in milliseconds between subsequent repetitions of the alarm. operation PendingIntent: Action to perform when the alarm goes off; typically comes from PendingIntent #getBroadcast. void setTime(long millis) Set the system wall clock time. Requires the permission android.permission.SET_TIME. Required Required Parameters millis long: time in milliseconds since the Epoch public void setTimeZone(String timeZone) Set the system's persistent default time zone. This is the time zone for all apps, even after a reboot. Use TimeZone.setDefault(TimeZone) if you just want to change the time zone within your app, and even then prefer to pass an explicit TimeZone to APIs that require it rather than changing the time zone for all threads. On Android M and above, it's a mistake to pass in a non-Olson timezone to this feature. Note that this is a bad idea on all Android releases because POSIX and the TimeZone class have opposite interpretations of '+' and '-' in the same non-Olson ID. Required Manifest.permission.SET_TIME_ZONE public void setWindow(int type, long windowStartMillis, long windowLengthMillis, PendingIntent operation) Schedule an alarm to be rendered within a given window. This method is similar to set (int, long, android.app.PendingIntent), but allows the application to control exactly the degree to which its delivery can be customized by the OS. This method allows an application to take advantage of the battery optimizations arising from delivery batch, even when it has modest timeliness requirements for its alarms. This method can also be used to achieve strict order guarantees under various alarms by ensuring that the windows requested for each alarm do not cross. When exact delivery is not required, applications must use the default set (int, long, android.app.PendingIntent) method. This will give the OS the most flexibility to reduce wake and battery usage. For alarms to be delivered at exactly specified times with no acceptable variation, applications can use setExact(int, long, android.app.PendingIntent). Parameters type int: type alarm. Value is RTC_WAKEUP, RTC, ELAPSED_REALTIME_WAKEUP, or ELAPSED_REALTIME windowStartMillis long: The earliest time, in milliseconds, that the alarm should be delivered, expressed in the appropriate clock's units (depending on the alarm type). windowLengthMillis long: The length of the requested delivery window, in milliseconds. The alarm will be delivered no later than these many milliseconds to windowStartMillis. Note that this parameter is an expensive, not the timestamp of the end of the window. operation PendingIntent: Action to perform when the alarm goes off; typically comes from PendingIntent #getBroadcast. public void setWindow(int type, long windowStartMillis, long windowLengthMillis, String tag, AlarmManager.OnAlarmListener listener, Handler targetHandler) Direct callback version of setWindow(int, long, android.app.PendingIntent). Rather than providing a PendingIntent that sent when the alarm time is reached, this variant provides an OnAlarmListener instance that will be invoked at that time. The OnAlarmListener OnAlarmListener #onAlarm() method will via the specified target handler, or on the application's main looper if null if the targetHandler parameter is passed. Parameter.