Notification & Alarms

Notification Manager

1. **Overview**
2. **Types of Notification.**
3. **Notification Manager**
4. **Setting up Notifications**
5. **Canceling Notifications**
6. **Pending Intent**
7. **Example: NotificationManager**

Overview

- A notification is a message you can display to the user outside of your application's normal UI
- When you tell the system to issue a notification, it first appears as an icon in the **notification area**
- To see the details of the notification, the user opens the **notification drawer**
- Both the notification area and the notification drawer are system-controlled areas that the user can view at any time.

Notification Manager

The NotificationManager is used to notify the notifications

- The Notification Manager is received from context using **getSystemService**

```
NotificationManager notificationManager = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
```

- To issue notification, the Notification object is passed to system using **NotificationManager.notify()**

Notification Object

- A Notification object defines the details of the notification message that is displayed in the status bar and "Notifications" window, and any other alert settings, such as sounds and blinking lights. Notifcation object is created using Notification.Builder
- **Notification Builder:**
- At bare minimum, a Builder object must include the following:
  - A small icon, set by setSmallIcon()
  - A title, set by setContentTitle()
  - Detail text, set by setContentText()
- Optional settings for the notification object include:
  - An alert sound
  - A PendingIntent for the activity to be fired when the notification is selected
  - A vibrate setting
  - A flashing LED setting
  - A ticker-text message for the status bar

Create Simple Notifications

1. Get a reference to the Notification Manager:

```
NotificationManager notiMgr = (NotificationManager) this.getSystemService(NOTIFICATION_SERVICE);
```

2. Instantiate the Notification Object using Notification.Builder

```
Notification.Builder notiBuild = new Notification.Builder(this);
notiBuild.setContentTitle("Notify Title");
notiBuild.setContentText("Notify Text World");
notiBuild.setSmallIcon(R.drawable.ic_launcher);
Notification notiObj = notiBuild.build() ;
```

3. Pass the Notification Object to the Notification Manager:

```
private static final int NOTIFICATION_ID = 1;
mNotificationManager.notify(NOTIFICATION_ID, notiObj);
```

That's it. Your user has now been notified.

** **NotificationCompat.Builder** is older compatibility version of Notification Builder

Notifications : Adding Sound, Vibrations

4. Define the Notification's Sound:
   - To access your raw resources you need to create the Uri as follows:
     - android.resource://[PACKAGE_NAME]/[RESOURCE_ID]
   - So the code could end up looking like that:
     - Uri sound = Uri.parse("android.resource://" + getPackageName() + "/" + R.raw.break_up_party);
     - notiBuild.setSound(sound);
5. To define vibration pattern, pass an array of *long* values to the *vibrate* field:
   - long[] vibrate = {0,100,200,300};
   - notiBuild.setVibrate(vibrate);

Notification Action Using Pending Intent

- A PendingIntent is a token that you give to another application (e.g. Notification Manager, Alarm Manager or other 3rd party applications), which allows this other application to use the permissions of your application to execute a predefined piece of code.
- To perform a broadcast via a pending intent so get a PendingIntent via the getBroadcast() method of the PendingIntent class.
- To perform an activity via an pending intent you receive the activity via PendingIntent.getActivity().

```
Intent notificationIntent = new Intent(this, MyClass.class);
PendingIntent contentIntent = PendingIntent.getActivity(this, 0, notificationIntent, 0);
 notiBuild.setContentIntent(contentintent);
```

More features

- You can add several more features to your notifications using Notification fields and flags. Some useful features include the following:
- **"FLAG_AUTO_CANCEL"** flag Add this to the *flags* field to automatically cancel the notification after it is selected from the Notifications window.
- **"FLAG_INSISTENT"** flag Add this to the *flags* field to repeat the audio until the user responds.
- **"FLAG_ONGOING_EVENT**" flag Add this to the *flags* field to group the notification under the "Ongoing" title in the Notifications window. This indicates that the application is on-going — its processes is still running in the background, even when the application is not visible (such as with music or a phone call).
- **"FLAG_NO_CLEAR"** flag Add this to the *flags* field to indicate that the notification should *not* be cleared by the "Clear notifications" button. This is particularly useful if your notification is on-going.
- ***number* field** This value indicates the current number of events represented by the notification. The appropriate number is overlayed on top of the status bar icon. If you intend to use this field, then you must start with "1" when the Notification is first created. (If you change the value from zero to anything greater during an update, the number is not shown.)
- ***iconLevel* field** This value indicates the current level of a LevelListDrawable that is used for the notification icon. You can animate the icon in the status bar by changing this value to correlate with the drawable's defined in a LevelListDrawable. See the LevelListDrawable reference for more information. See the Notification class reference for more information about additional features that you can customize for your application.

Example: NotificationManager

- Create a new project called *com.sisoft.notificationmanager* with the *activity* class called CreateNotificationActivity. This *activity* should use the main.xml layout file.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical" >

 <Button android:id="@+id/button1" android:layout_width="match_parent"
android:layout_height="match_parent"
android:onClick="createNotification"
android:text="Create Notification" >
</Button>
</LinearLayout>
```

```
public class CreateNotificationActivity extends Activity
{ @Override
 public void onCreate(Bundle savedInstanceState)
{
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);
}
public void createNotification(View view)
{ // Prepare intent which is triggered if the // notification is selected
 Intent intent = new Intent(this, NotificationReceiverActivity.class);
PendingIntent pIntent = PendingIntent.getActivity(this, 0, intent, 0);
// Build notification // Actions are just fake
Notification noti = new Notification.Builder(this)
 .setContentTitle("New mail from " + "test@gmail.com")
 .setContentText("Subject")
.setSmallIcon(R.drawable.icon)
.setContentIntent(pIntent)
.addAction(R.drawable.icon, "Call", pIntent)
.addAction(R.drawable.icon, "More", pIntent) .addAction(R.drawable.icon, "And more",
pIntent).build(); NotificationManager notificationManager = (NotificationManager)
getSystemService(NOTIFICATION_SERVICE);
 // Hide the notification after its selected
 noti.flags |= Notification.FLAG_AUTO_CANCEL;
 notificationManager.notify(0, noti); } }
```

Manage Notification

When you need to issue a notification multiple times for the same type of event, you should avoid making a completely new notification. Instead, you should consider updating a previous notification, either by changing some of its values or by adding to it, or both.

The following section describes how to update notifications and also how to remove them.

- Modify Notification
- Delete/Remove Notification

Modify a Notification

- To set up a notification so it can be updated, issue it with a notification ID by calling NotificationManager.notify(ID, notification)

- To update this notification once you've issued it, update or create a Notification.Builder object, build a Notification object from it, and issue the Notification with the same ID you used previously

- The following snippet demonstrates a notification that is updated to reflect the number of events that have occurred. It stacks the notification, showing a summary

Modify a Notification

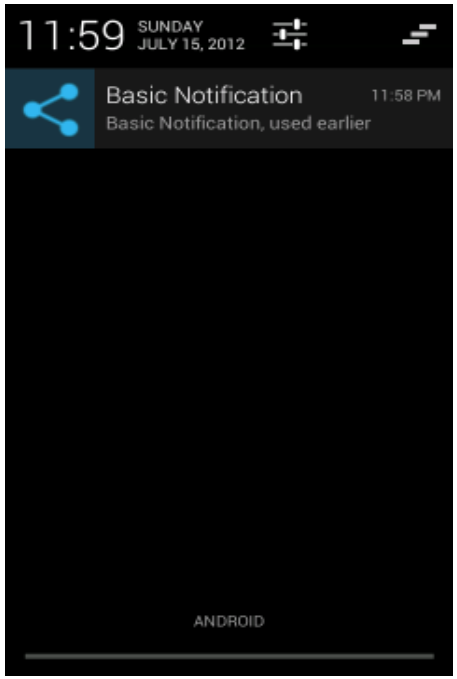
Example :

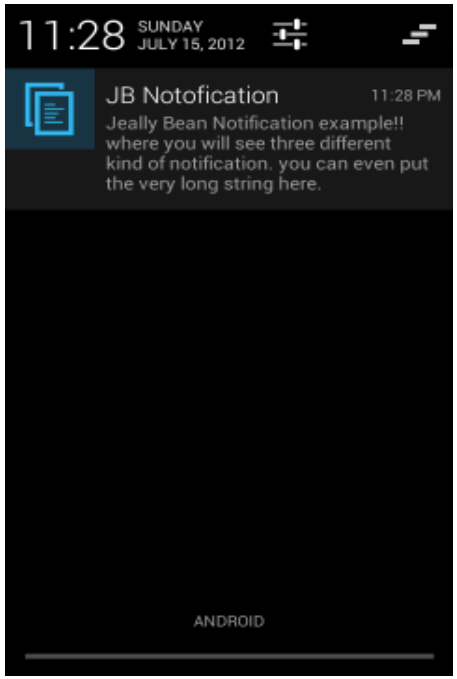
```
mNotificationManager =
      (NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE);
// Sets an ID for the notification, so it can be updated
int notifyID = 1;
mNotifyBuilder = new NotificationCompat.Builder(this)
    .setContentTitle("New Message")
    .setContentText("You've received new messages.")
    .setSmallIcon(R.drawable.ic_notify_status)
numMessages = 0;
// Start of a loop that processes data and then notifies the user
...
    mNotifyBuilder.setContentText(currentText)
        .setNumber(++numMessages);
    // Because the ID remains unchanged, the existing notification is
    // updated.
    mNotificationManager.notify(
          notifyID,
          mNotifyBuilder.build());
...
```

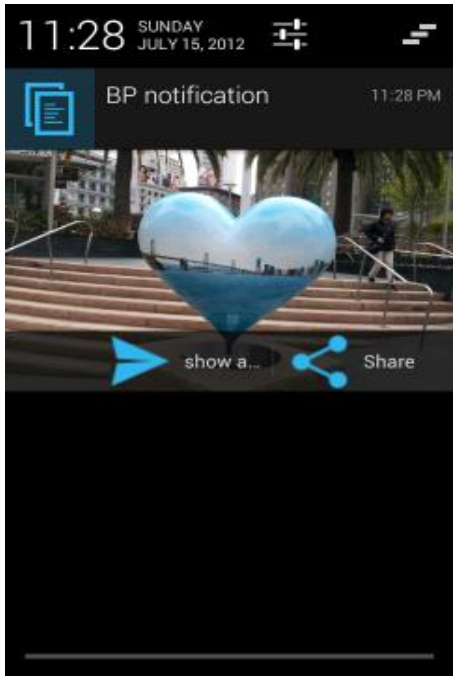
Remove Notifications

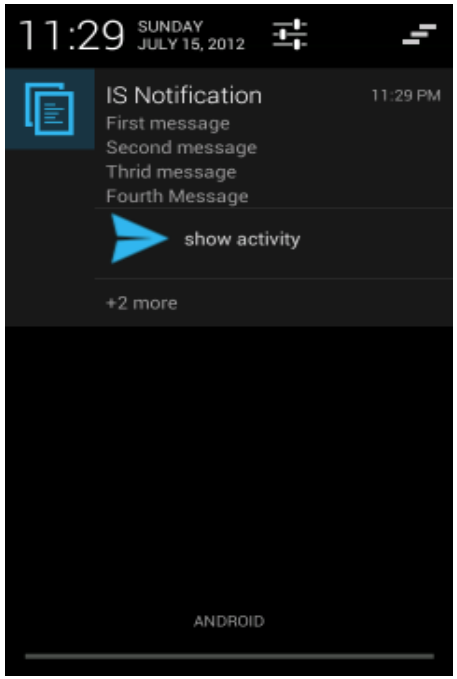
- Notifications remain visible until one of the following happens:
- The user dismisses the notification either individually or by using "Clear All" (if the notification can be cleared).
- The user touches the notification, and you called setAutoCancel() when you created the notification.
- You call cancel() for a specific notification ID. This method also deletes ongoing notifications.
- You call cancelAll(), which removes all of the notifications you previously issued.

Notification Display Styles









Notification Display Styles

- You may have heard about Android Jelly Bean (API level 16). Google has improved a lot of features and introduced new features. One of them is the notification. Now they have made the notification more versatile by introducing media rich notification.
- Google has come up with three special style of notification which are mentioned below. Even developer can write his own customized notification style using remote view .
- The old Notification class constructor has been deprecated and a brand new and enhanced version of Notification has been introduced.

Notification Display Styles

- Notifications in the notification drawer can appear in one of two visual styles, depending on the version and the state of the drawer:

1. Normal view :

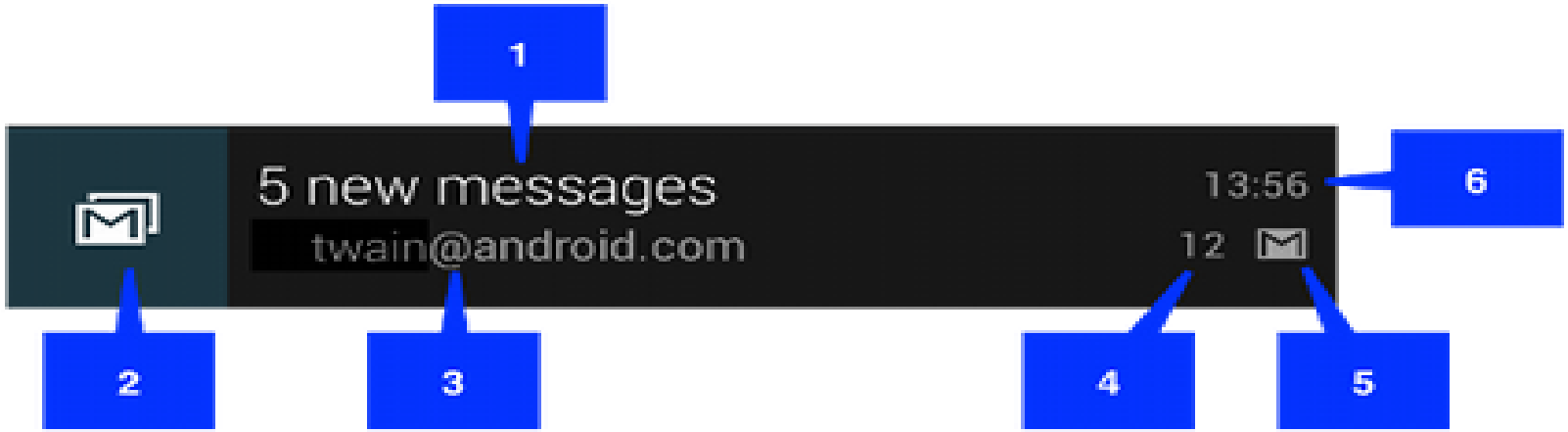
The standard view of the notifications in the notification drawer.

2. Big view :

A large view that's visible when the notification is expanded. Big view is part of the expanded notification feature available as of Android 4.1

Normal view

- A notification in normal view appears in an area that's up to 64 dp tall. Even if you create a notification with a big view style, it will appear in normal view until it's expanded. This is an example of a normal view:



Normal view

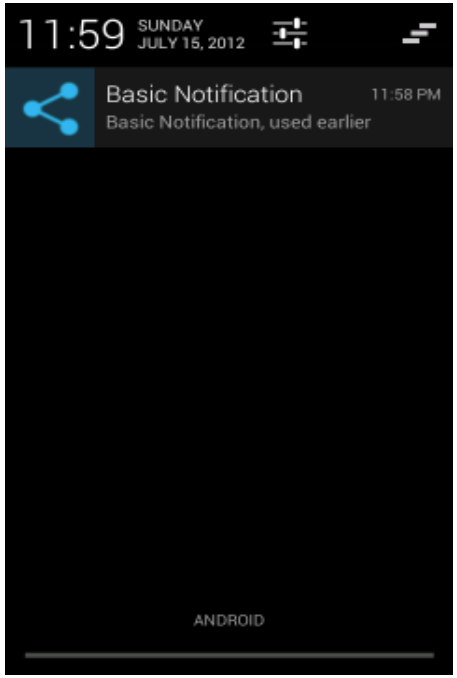
The callouts in the illustration refer to the following:

1. Content title
2. Large icon
3. Content text
4. Content info
5. Small icon
6. Time that the notification was issued. You can set an explicit value with setWhen(); if you don't it defaults to the time that the system received the notification.

1. Normal view

- **Normal Notification** – Shows simple and short notification with icon.

```
public void sendBasicNotification(View view) {
 Notification notification = new
Notification.Builder(this)
  .setContentTitle("Basic Notification")
  .setContentText("Basic Notification, used earlier")
  .setSmallIcon(R.drawable.ic_launcher_share).build();
 notification.flags |= Notification.FLAG_AUTO_CANCEL;
 NotificationManager notificationManager =
getNotificationManager();
 notificationManager.notify(0, notification);
 }
```



2. Big view

- A notification's big view appears only when the notification is expanded, which happens when the notification is at the top of the notification drawer, or when the user expands the notification with a gesture. Expanded notifications are available starting with Android 4.1.
- The following screenshot shows an inbox-style notification:

2. Big view

Notice that the big view shares most of its visual elements with the normal view. The only difference is callout number 7, the details area. Each big view style sets this area in a different way. The available styles are:

- Big picture style :- The details area contains a bitmap up to 256 dp tall in its detail section.
- Big text style :- Displays a large text block in the details section.
- Inbox style :- Displays lines of text in the details section.

All of the big view styles also have the following content options that aren't available in normal view:

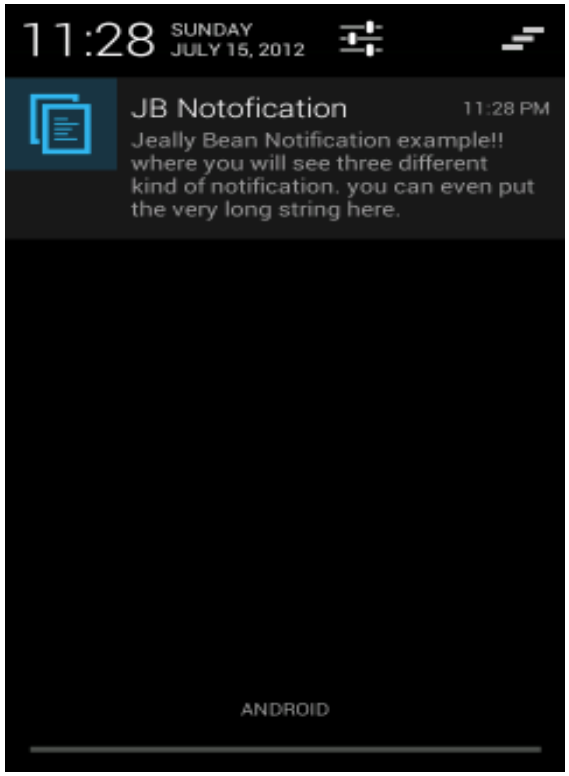
- Big content title Allows you to override the normal view's content title with a title that appears only in the expanded view.
- Summary text Allows you to add a line of text below the details area.

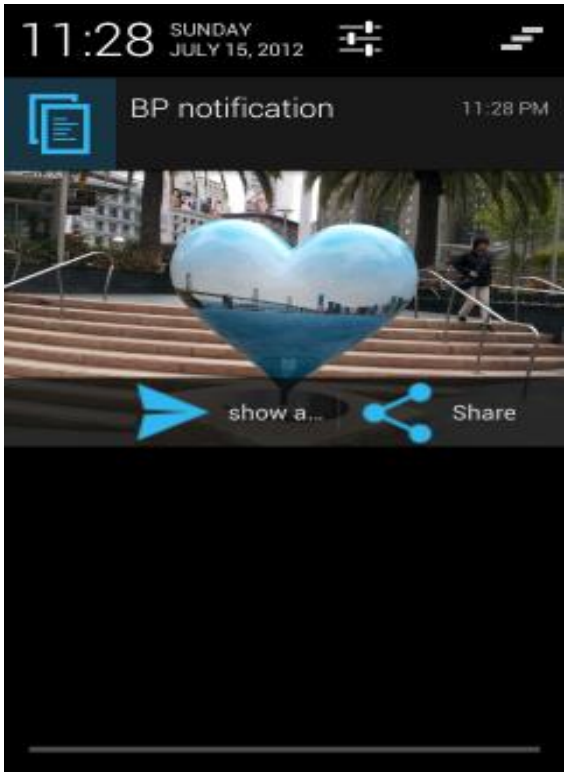
2. Big Notification

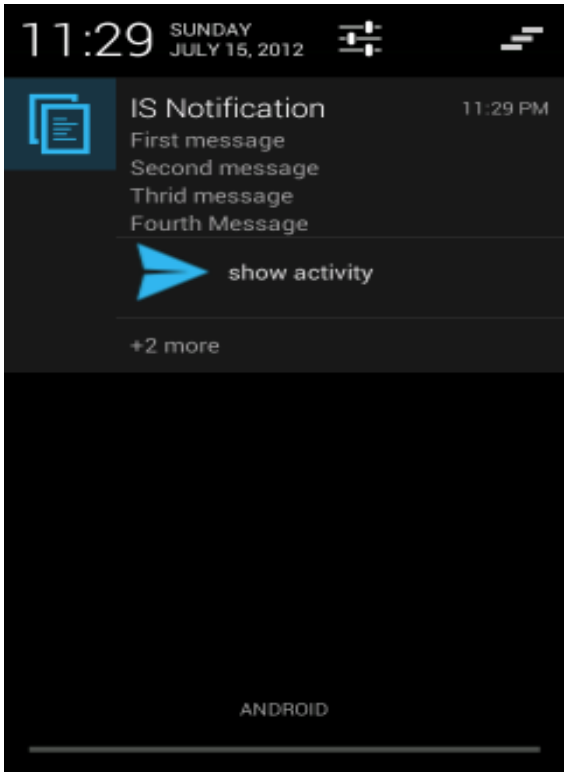
- ***Notification.Builder*** class has been introduced to make this task easier. This class returns the builder object which is configurable according to your requirements.
- The helper classes have been introduced like
  - ***Notification.BigPictureStyle,***
  - ***Notification.BigTextStyle,***
  - ***Notification.InboxStyle***.
- These classes are re-builder classes which take object created by Notification.Builder class and modify the behavior

2. Big Notification

- **Big Picture Notification** – Shows visual content such as bitmap.
- **Big Text Notification** – Shows multiline Textview object.
- **Inbox Style Notification –** Shows any kind of list, e.g messages, headline etc.







(a)BigTextStyleNotification

```
public void sendBigTextStyleNotification(View view) {
 String msgText = "Jeally Bean Notification example!! "
  + "where you will see three different kind of notification. "
  + "you can even put the very long string here.";

 NotificationManager notificationManager = getNotificationManager();
 PendingIntent pi = getPendingIntent();
 Builder builder = new Notification.Builder(this);
 builder.setContentTitle("Big text Notofication")
  .setContentText("Big text Notification")
  .setSmallIcon(R.drawable.ic_launcher)
  .setAutoCancel(true);
  .setPriority(Notification.PRIORITY_HIGH)
  .addAction(R.drawable.ic_launcher_web, "show activity", pi);
 Notification notification = new Notification.BigTextStyle(builder)
  .bigText(msgText).build();

 notificationManager.notify(0, notification);
}
```

(b)BigPictureStyleNotification

```
public void sendBigPictureStyleNotification(View view) {
 PendingIntent pi = getPendingIntent();
 Builder builder = new Notification.Builder(this);
 builder.setContentTitle("BP notification")
  // Notification title
  .setContentText("BigPicutre notification")
  // you can put subject line.
  .setSmallIcon(R.drawable.ic_launcher)
  // Set your notification icon here.
  .addAction(R.drawable.ic_launcher_web, "show activity", pi)
  .addAction(
   R.drawable.ic_launcher_share,
   "Share",
   PendingIntent.getActivity(getApplicationContext(), 0,
    getIntent(), 0, null));

 // Now create the Big picture notification.
 Notification notification = new Notification.BigPictureStyle(builder)
  .bigPicture(
   BitmapFactory.decodeResource(getResources(),
    R.drawable.big_picture)).build();
 // Put the auto cancel notification flag
 notification.flags |= Notification.FLAG_AUTO_CANCEL;
 NotificationManager notificationManager = getNotificationManager();
 notificationManager.notify(0, notification);
}
```

(c)InboxStyleNotification

```
public void sendInboxStyleNotification(View view)
{
 PendingIntent pi = getPendingIntent();
 Builder builder = new Notification.Builder(this)
  .setContentTitle("IS Notification")
  .setContentText("Inbox Style notification!!")
  .setSmallIcon(R.drawable.ic_launcher)
  .addAction(R.drawable.ic_launcher_web, "show activity", pi);

 Notification notification = new Notification.InboxStyle(builder)
  .addLine("First message").addLine("Second message")
  .addLine("Thrid message").addLine("Fourth Message")
  .setSummaryText("+2 more").build();
 // Put the auto cancel notification flag
 notification.flags |= Notification.FLAG_AUTO_CANCEL;
 NotificationManager notificationManager =
getNotificationManager();
 notificationManager.notify(0, notification);
 }
```

Alarm Manager

AlarmManager

- Alarms (based on the AlarmManager class) give you a way to perform **time-based** operations **outside the lifetime** of your application.
- It is recommended to use AlarmManager when you want your application code to be run at a specific time, even if your application is not currently running. For other timing operation handler should be used because it is easy to use. Handler is covered in other tutorial
- AlarmManager has access to the system alarm services. With the help of AlarmManager you can schedule execution of code in future.
- AlarmManager object can't instantiate directly however it can be retrieved by calling ***Context.getSystemService(Context.ALARM_SERVICE)***.
- AlarmManager is always registered with intent. When an alarm goes off, the Intent which has been registered with AlarmManager, is broadcasted by the system automatically. This intent starts the target application if it is not running.

AlarmManager: Alarm Type

There are two general clock types for alarms

- Elapsed real time: This uses the "time since system boot" as a reference. This means that elapsed real time is suited to setting an alarm based on the passage of time (for example, an alarm that fires every 30 seconds) since it isn't affected by time zone/locale.
- Real time clock(RTC): Real time clock uses UTC (wall clock) time. The real time clock type is better suited for alarms that are dependent on current locale.
- Both types have a "wakeup" version, which says to wake up the device's CPU if the screen is off. This ensures that the alarm will fire at the scheduled time.

AlarmManager: Alarm Type

- 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.

AlarmManager

Methods	Description
Set	**public void set (int type, long triggerAtMillis, PendingIntent operation)**
setExact	Schedule an alarm to be delivered precisely at the stated time. **public void setExact (int type, long triggerAtMillis, PendingIntent operation)**
setInexactRepeating	**public void setInexactRepeating (int type, long triggerAtMillis, long intervalMillis, PendingIntent operation)** **public void setInexactRepeating (int type, long triggerAtMillis, long intervalMillis, PendingIntent operation)**
setRepeating	Schedule a repeating alarm. **public void setRepeating (int type, long triggerAtMillis, long intervalMillis, PendingIntent operation)**

Example - AlarmManager

- We will create one-time timer and the repeating timer, and also to cancel the repeating timer. Here timer and alarm have been used interchangeably, but in this tutorial context both of them have the same meaning.

Example - AlarmManager

- We are going to define the BroadcastReciever which handles the intent registered with AlarmManager.
- In the given class onReceive() method has been defined. This method gets invoked as soon as intent is received. Once we receive the intent we try to get the extra parameter associated with this intent.
- This extra parameter is user-defined i.e ONE_TIME, basically indicates whether this intent was associated with one-time timer or the repeating one.
- Once the ONE_TIME parameter value has been extracted, Toast message is displayed accordingly. Helper methods have also been defined, which can be used from other places with the help of objects i.e **setAlarm(), cancelAlarm() and onetimeTimer()** methods.
- These methods can also be defined somewhere else to do operation on the timer i.e set, cancel, etc. To keep this tutorial simple, we have defined it in BroadcastReceiver.

Example - AlarmManager

- **setAlarm():** This method sets the repeating alarm by use of setRepeating() method. setRepeating() method needs four arguments:
  1. type of alarm,
  2. trigger time: set it to the current time
  3. interval in milliseconds: in this example we are passing 5 seconds (1000 * 5 milliseconds)
  4. pending intent: It will get registered with this alarm. When the alarm gets triggered the pendingIntent will be broadcasted.
- **cancelAlarm()**: This method cancels the previously registered alarm by calling the cancel() method. cancel() method takes pendingIntent as an argument. The pendingIntent should be matching one, only then the cancel() method can remove the alarm from the system.

Example - AlarmManager

- **onetimeTimer():** This method creates an one-time alarm. This can be achieved by calling set() method. set() method takes three arguments:
  1. type of alarm
  2. trigger time
  3. pending intent

Example - AlarmManager

```
public class AlarmManagerBroadcastReceiver extends BroadcastReceiver {

final public static String ONE_TIME = "onetime";

@Override
public void onReceive(Context context, Intent intent) {
 PowerManager pm = (PowerManager) context.getSystemService(Context.POWER_SERVICE);
    PowerManager.WakeLock wl = pm.newWakeLock(PowerManager.PARTIAL_WAKE_LOCK, "YOUR TAG");
    //Acquire the lock
    wl.acquire();

    //You can do the processing here.
    Bundle extras = intent.getExtras();
    StringBuilder msgStr = new StringBuilder();

    if(extras != null && extras.getBoolean(ONE_TIME, Boolean.FALSE)){
     //Make sure this intent has been sent by the one-time timer button.
     msgStr.append("One time Timer : ");
    }
    Format formatter = new SimpleDateFormat("hh:mm:ss a");
    msgStr.append(formatter.format(new Date()));

    Toast.makeText(context, msgStr, Toast.LENGTH_LONG).show();

    //Release the lock
    wl.release();
}
```

Example - AlarmManager

```
public void SetAlarm(Context context)
 {
   AlarmManager am=(AlarmManager)context.getSystemService(Context.ALARM_SERVICE);
   Intent intent = new Intent(context, AlarmManagerBroadcastReceiver.class);
   intent.putExtra(ONE_TIME, Boolean.FALSE);
   PendingIntent pi = PendingIntent.getBroadcast(context, 0, intent, 0);
   //After after 5 seconds
   am.setRepeating(AlarmManager.RTC_WAKEUP, System.currentTimeMillis(), 1000 * 5 , pi);
 }

 public void CancelAlarm(Context context)
 {
   Intent intent = new Intent(context, AlarmManagerBroadcastReceiver.class);
   PendingIntent sender = PendingIntent.getBroadcast(context, 0, intent, 0);
   AlarmManager alarmManager = (AlarmManager)
context.getSystemService(Context.ALARM_SERVICE);
   alarmManager.cancel(sender);
 }

 public void setOnetimeTimer(Context context){
  AlarmManager am=(AlarmManager)context.getSystemService(Context.ALARM_SERVICE);
   Intent intent = new Intent(context, AlarmManagerBroadcastReceiver.class);
   intent.putExtra(ONE_TIME, Boolean.TRUE);
   PendingIntent pi = PendingIntent.getBroadcast(context, 0, intent, 0);
   am.set(AlarmManager.RTC_WAKEUP, System.currentTimeMillis(), pi);
 }
}
```

Example

- Given below is the manifest file. Here, WAKE_LOCK permission is required because the wake lock is being used while processing in onReceive() method present in AlarmManagerBroadcastReceiver class. AlarmManagerBroadcastReceiver has been registered as broadcast receiver.

```
<manifest android:versioncode="1" android:versionname="1.0"
   package="com.rakesh.alarmmanagerexample"
   xmlns:android="http://schemas.android.com/apk/res/android">

 <uses-sdk android:minsdkversion="10" android:targetsdkversion="15"/>
 <uses-permission android:name="android.permission.WAKE_LOCK"/>
  <application android:icon="@drawable/ic_launcher"
   android:label="@string/app_name" android:theme="@style/AppTheme">
   <activity android:label="@string/title_activity_alarm_manager"
    android:name="com.rakesh.alarmmanagerexample.AlarmManagerActivity">
     <intent-filter>
       <action android:name="android.intent.action.MAIN"/>
       <category android:name="android.intent.category.LAUNCHER" />
     </intent-filter>
   </activity>
   <receiver
android:name="com.rakesh.alarmmanagerexample.AlarmManagerBroadcastReceiver">
   </receiver>
  </application>
</manifest>
```

Example

Now let's define the activity class which defines some methods. These methods are going to handle the button clicks. Here in this class we create an instance of AlarmManagerBroadcastReciever which will help us to access setAlarm(), cancelAlarm() and setOnetime(). Rest of the code is easy to understand.

```
public class AlarmManagerActivity extends Activity {

private AlarmManagerBroadcastReceiver alarm;
 @Override
 public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_alarm_manager);
   alarm = new AlarmManagerBroadcastReceiver();
 }
 @Override
protected void onStart() { super.onStart(); }

 public void startRepeatingTimer(View view) {
 Context context = this.getApplicationContext();
 if(alarm != null){
  alarm.SetAlarm(context);
 }else{
  Toast.makeText(context, "Alarm is null", Toast.LENGTH_SHORT).show();
 }
 }
```

JUNKEES

AlarmManager

Methods:

Method	Description
set()	Schedules an alarm for one time.
setInexactRepeating()	Schedules an alarm with inexact repeating. Trigger time doesn't follow any strict restriction.
setRepeating()	Schedules an alarm with exact repeating time.
setTime()	Sets the system's wall clock time.
setTimeZone()	Sets the system's default time zone.

Example

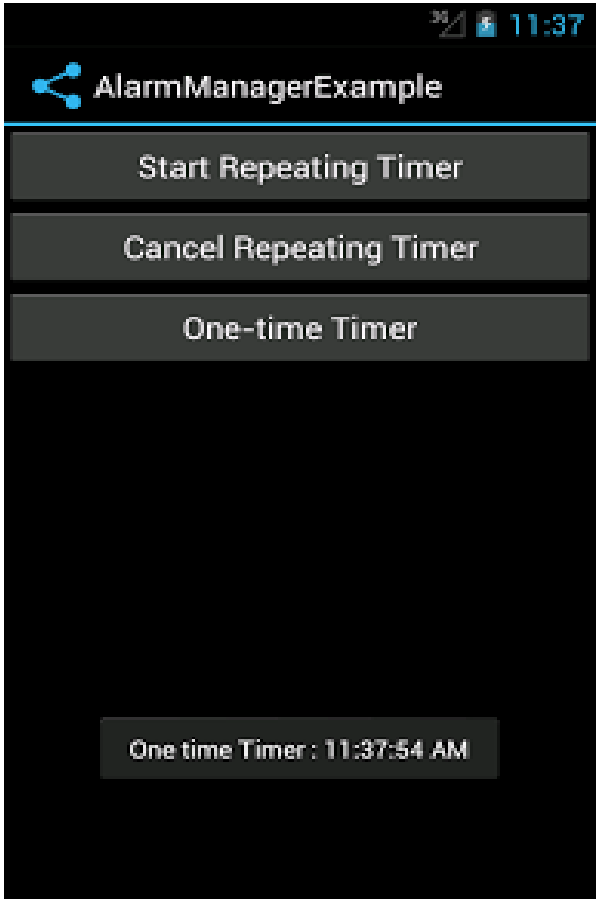
```
public void cancelRepeatingTimer(View view){
  Context context = this.getApplicationContext();
  if(alarm != null){
   alarm.CancelAlarm(context);
  }else{
   Toast.makeText(context, "Alarm is null", Toast.LENGTH_SHORT).show();
  }
 }

public void onetimeTimer(View view)
{
  Context context = this.getApplicationContext();
  if(alarm != null)
   { alarm.setOnetimeTimer(context); }
  else{
   Toast.makeText(context, "Alarm is null", Toast.LENGTH_SHORT).show();
  }
 }
```

Once you are done with the coding, just execute the project and you will find the similar kind of application running in your emulator.

Run


11:08
APPS
WIDGETS
AlarmManagerExample
API Demos
Browser
Calculator
Calendar
Camera
Clock
Custom Locale
Dev Tools
Downloads
Email
Gallery
Gestures Builder
Latitude
Local
Maps
Messaging
Music
Navigation
People


11:37
AlarmManagerExample
Start Repeating Timer
Cancel Repeating Timer
One-time Timer
One time Timer : 11:37:54 AM