Notifications

Notifications

Mechanism for notifying a user about an event.
Handled by Notification Manager
Allows invisible components to alert users that events have occurred that may require attention

What is a notification?

A notification is a message that Android displays outside your app's UI to provide the user with reminders, communication from other people, or other timely information from your app.

Users can tap the notification to open your app or take an action directly from the notification.

Figure 1. Notifications in the notification area.

Figure 2. Notifications in the notification drawer.

Appearances on a device

Notifications appear to users in different locations and formats, such as an icon in the status bar, a more detailed entry in the notification drawer, as a badge on the app's icon etc.

A notification remains visible in the notification drawer until dismissed by the app or the user.

Figure 1. Notification icons appear on the left side of the status bar

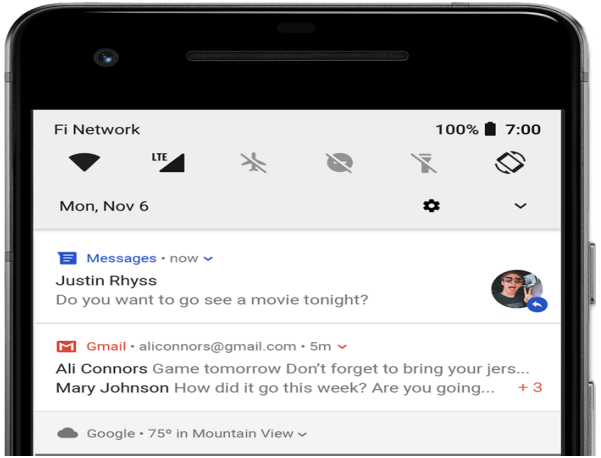
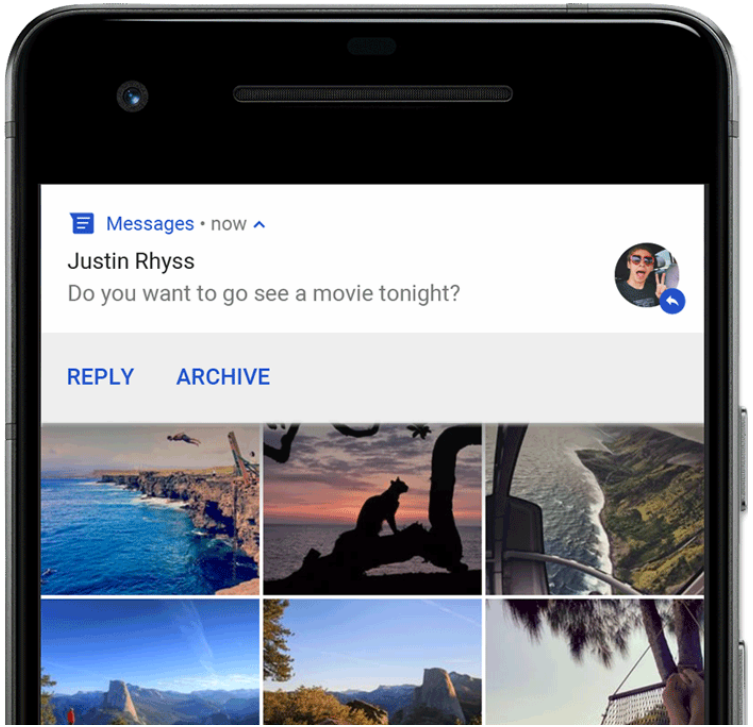


Figure 2. Notifications in the notification drawer

Heads-up notification

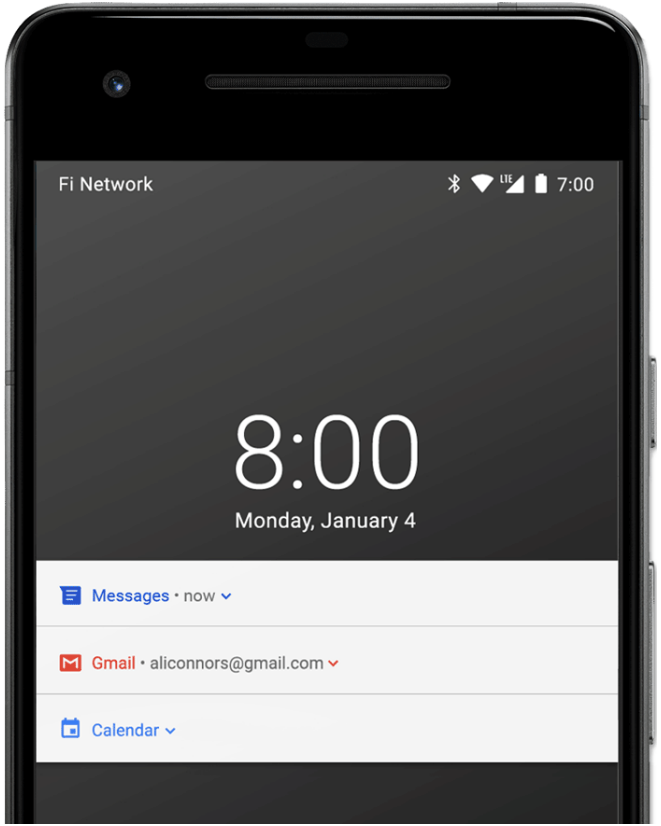
Beginning with Android 5.0, notifications can briefly appear in a floating window called a *heads-up notification*.

This behavior is normally for important notifications that the user should know about immediately, and it appears only if the device is unlocked.



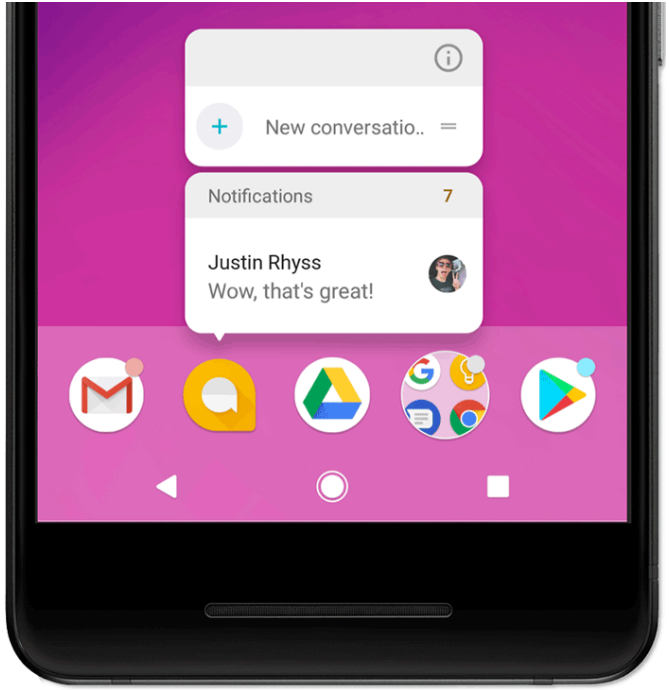
Lock screen notification

Beginning with Android 5.0, notifications can appear on the lock screen.
You can programmatically set the level of detail visible in notifications posted by your app on a secure lock screen, or even whether the notification will show on the lock screen at all.
Users can use the system settings to choose the level of detail visible in lock screen notifications, including the option to disable all lock screen notifications.



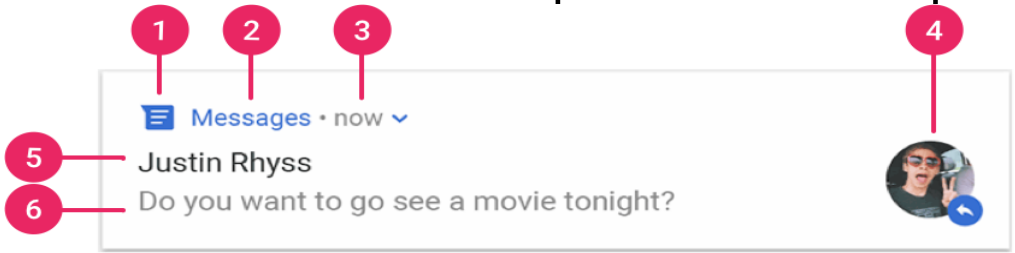
App icon badge notification

In supported launchers on devices running Android 8.0 (API level 26) and higher, app icons indicate new notifications with a colored "badge" (also known as a "notification dot") on the corresponding app launcher icon.
Users can long-press on an app icon to see the notifications for that app. Users can then dismiss or act on notifications from that menu, similar to the notification drawer.



Notification anatomy

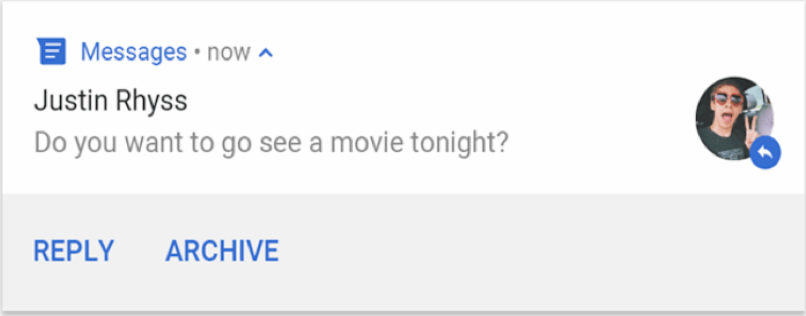
The design of a notification is determined by system templates—your app simply defines the contents for each portion of the template.



1. **Small icon:** This is required and set with setSmallIcon().
2. **App name:** This is provided by the system.
3. **Time stamp:** This is provided by the system but you can override with setWhen() or hide it with setShowWhen(false).
4. **Large icon:** This is optional (usually used only for contact photos; do not use it for your app icon) and set with setLargeIcon().
5. **Title:** This is optional and set with setContentTitle().
6. **Text:** This is optional and set with setContentText().

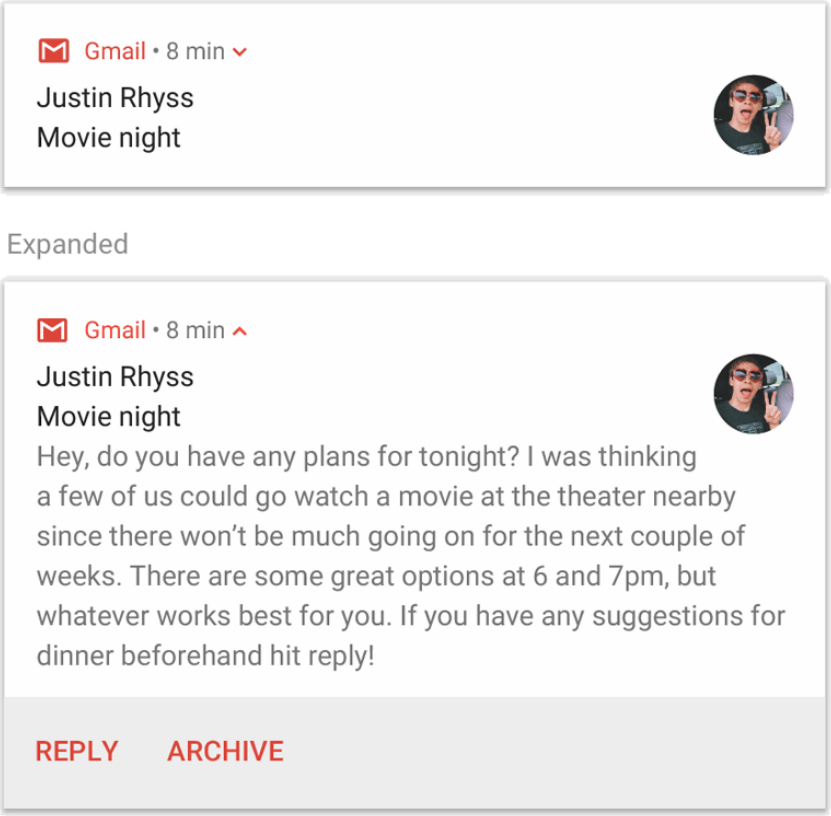
Notification actions

Although it's not required, every notification should open an appropriate app activity when tapped. In addition to this default notification action, you can add action buttons that complete an app-related task from the notification (often without opening an activity)
Starting in Android 7.0 (API level 24), you can also add an action to reply to messages or enter other text directly from the notification.



Expandable notification

By default, the notification's text content is truncated to fit on one line. If you want your notification to be longer, you can enable a larger text area that's expandable by applying an additional template.
You can also create an expandable notification with an image, in inbox style, a chat conversation, or media playback controls.

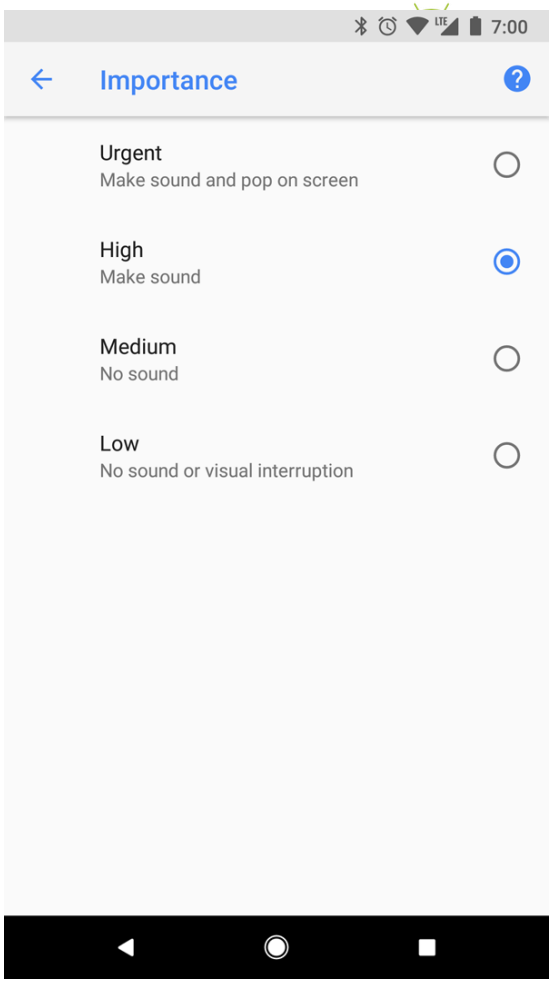


Notification importance

Android uses the *importance* of a notification to determine how much the notification should interrupt the user (visually and audibly). The higher the importance of a notification, the more interruptive the notification will be.

The possible importance levels are the following:

- **Urgent**: Makes a sound and appears as a heads-up notification.
- **High**: Makes a sound.
- **Medium**: No sound.
- **Low**: No sound and does not appear in the status bar.



Create a Notification

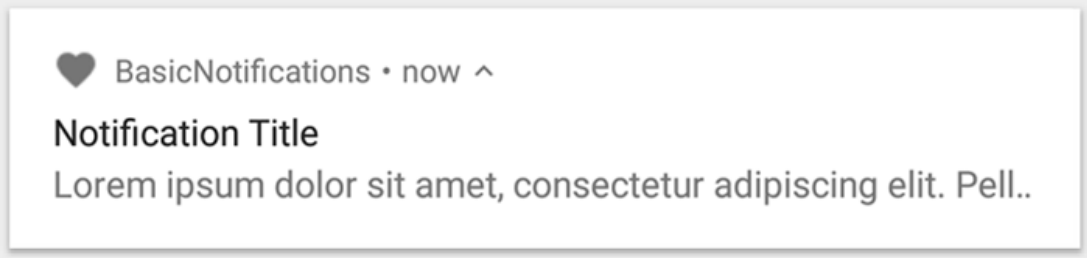
Add the support library

Although most projects created with Android Studio include the necessary dependencies to use NotificationCompat, you should verify that your module-level build.gradle file includes the following dependency:

```
dependencies {
    implementation "com.android.support:support-compat:27.1.1"
}
```

Create a basic notification

A notification in its most basic and compact form (also known as collapsed form) displays an icon, a title, and a small amount of content text.




```
NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(this, CHANNEL_ID)
        .setSmallIcon(R.drawable.notification_icon)
        .setContentTitle(textTitle)
        .setContentText(textContent)
        .setPriority(NotificationCompat.PRIORITY_DEFAULT);
```

Notice that the NotificationCompat.Builder constructor requires that you provide a channel ID. This is required for compatibility with Android 8.0 (API level 26) and higher, but is ignored by older versions.

Create a basic notification

By default, the notification's text content is truncated to fit one line. If you want your notification to be longer, you can enable an expandable notification by adding a style template with setStyle(). For example, the following code creates a larger text area:

```
NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(this, CHANNEL_ID)
        .setSmallIcon(R.drawable.notification_icon)
        .setContentTitle("My notification")
        .setContentText("Much longer text that cannot fit one line...")
        .setStyle(new NotificationCompat.BigTextStyle()
                .bigText("Much longer text that cannot fit one line..."))
        .setPriority(NotificationCompat.PRIORITY_DEFAULT);
```

Set the notification's tap action

Every notification should respond to a tap, usually to open an activity in your app that corresponds to the notification. To do so, you must specify a content intent defined with a PendingIntent object and pass it to setContentIntent().

```
// Create an explicit intent for an Activity in your app
Intent intent = new Intent(this, AlertDetails.class);
intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TASK);
PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, intent, 0);

NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(this, CHANNEL_ID)
        .setSmallIcon(R.drawable.notification_icon)
        .setContentTitle("My notification")
        .setContentText("Hello World!")
        .setPriority(NotificationCompat.PRIORITY_DEFAULT)
        // Set the intent that will fire when the user taps the notification
        .setContentIntent(pendingIntent)
        .setAutoCancel(true);
```

Notice this code calls setAutoCancel(), which automatically removes the notification when the user taps it.

Show the notification

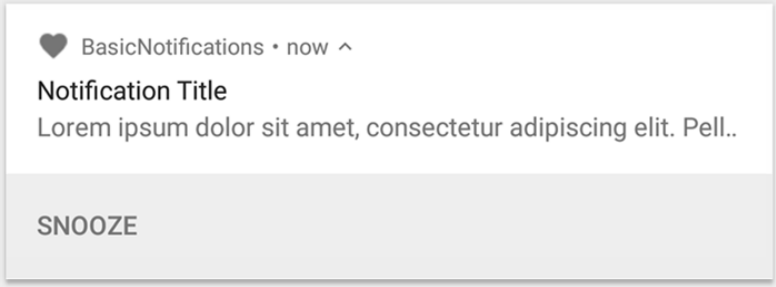
To make the notification appear, call NotificationManagerCompat.notify(), passing it a unique ID for the notification and the result of NotificationCompat.Builder.build(). For example:

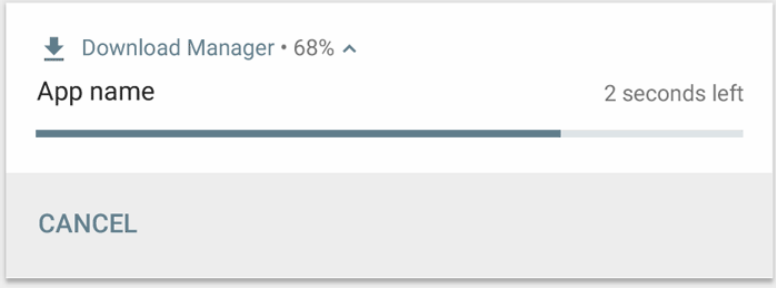
```
NotificationManagerCompat notificationManager = NotificationManagerCompat.from(this);

// notificationId is a unique int for each notification that you must define
notificationManager.notify(notificationId, mBuilder.build());
```

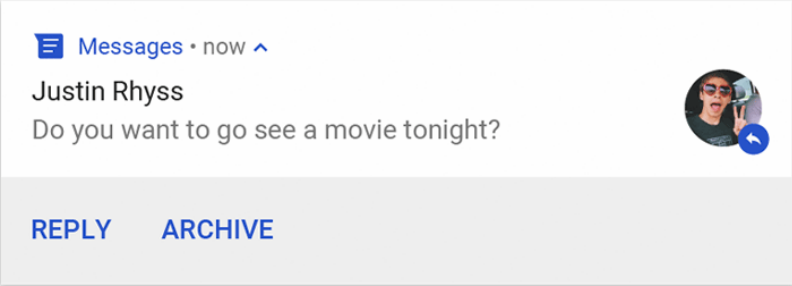
Remember to save the **notification ID** that you pass to NotificationManagerCompat.notify() because you'll need it later if you want to **update** or **remove** the notification.

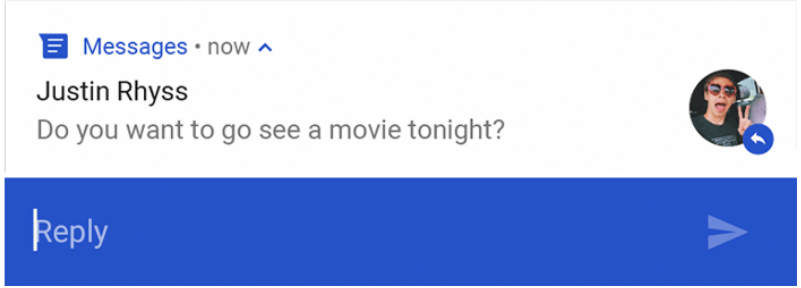
Other Features

Add action buttons
Add a progress bar
Add a direct reply action









Remove Notifications

Notifications remain visible until one of the following happens:

- The user dismisses the notification.
- 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.
- If you set a timeout when creating a notification using setTimeoutAfter(), the system cancels the notification after the specified duration elapses.

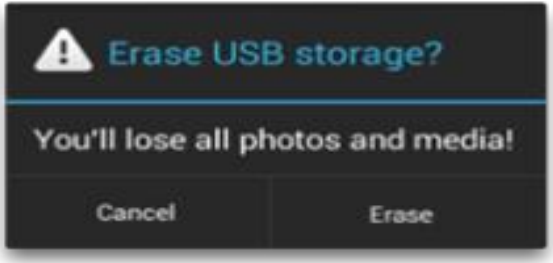
Reference

https://developer.android.com/guide/topics/ui/notifiers/notifications
https://developer.android.com/training/notify-user/build-notification

Dialogs

Dialogs

A dialog is a small window that prompts the user to make a decision or enter additional information.



AlertDialog

The AlertDialog class allows you to build a variety of dialog designs and is often the only dialog class you'll need.

1. **Title -** This is optional and should be used only when the content area is occupied by a detailed message, a list, or custom layout. If you need to state a simple message or question, you don't need a title.

2. **Content area -** This can display a message, a list, or other custom layout.

3. **Action buttons -** There should be no more than three action buttons in a dialog.

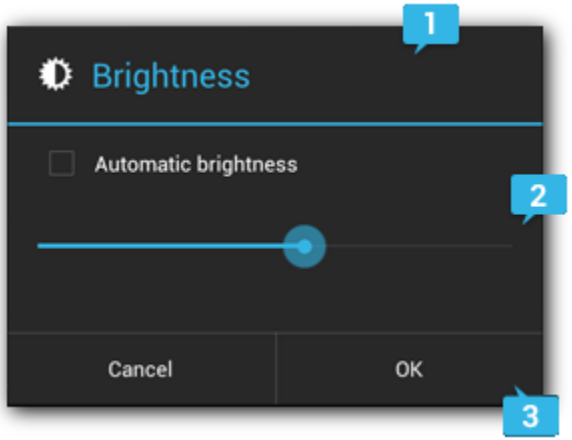
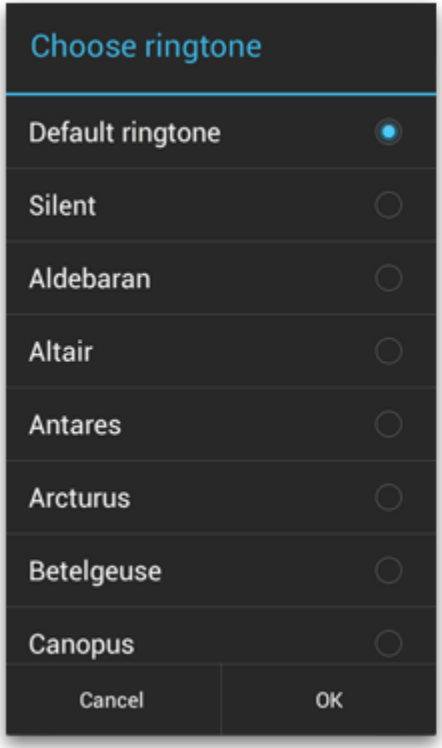
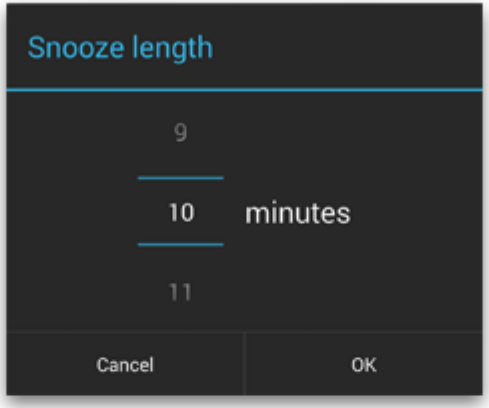
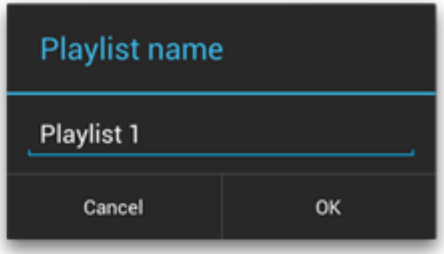


Figure 1. A dialog with a message and two action buttons.

More Examples

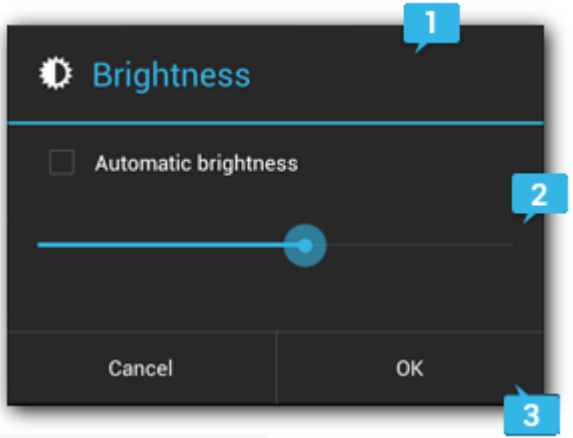






AlertDialog

The AlertDialog.Builder class provides APIs that allow you to create an AlertDialog with these kinds of content, including a custom layout.



To build an AlertDialog:

```
// 1. Instantiate an AlertDialog.Builder with its constructor
AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());

// 2. Chain together various setter methods to set the dialog characteristics
builder.setMessage(R.string.dialog_message)
       .setTitle(R.string.dialog_title);

// 3. Get the AlertDialog from create()
AlertDialog dialog = builder.create();
```

DialogFragment

- You should use a DialogFragment as a container for your dialog.
- Typically will return an AlertDialog.

DialogFragment

```java
public class FireMissilesDialogFragment extends DialogFragment {
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // Use the Builder class for convenient dialog construction
        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
        builder.setMessage(R.string.dialog_fire_missiles)
               .setPositiveButton(R.string.fire, new DialogInterface.OnClickListener() {
                   public void onClick(DialogInterface dialog, int id) {
                       // FIRE ZE MISSILES!
                   }
               })
               .setNegativeButton(R.string.cancel, new DialogInterface.OnClickListener() {
                   public void onClick(DialogInterface dialog, int id) {
                       // User cancelled the dialog
                   }
               });
        // Create the AlertDialog object and return it
        return builder.create();
    }
}
```

MainActivity.java

```
Button button = (Button) findViewById(R.id.button);
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        FireMissilesDialogFragment fragment = new FireMissilesDialogFragment();
        fragment.show(getSupportFragmentManager(), "missiles");
    }
});
```

Now, when you create an instance of this class and call show() on that object, the dialog appears as shown in figure.

The second argument, "missiles", is a unique tag name that the system uses to save and restore the fragment state when necessary.

AlertDialog Customizability

Adding a list

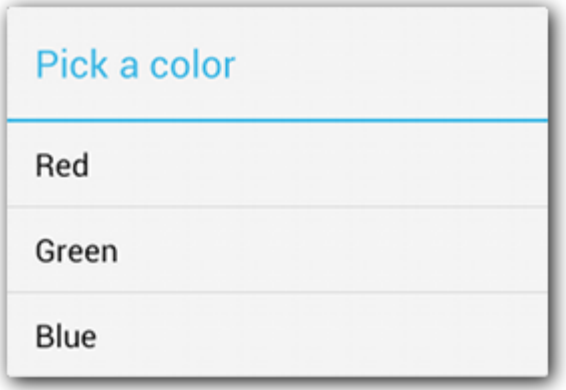


Figure 1. A dialog with a title and list.

Adding a persistent multiple-choice or single-choice list

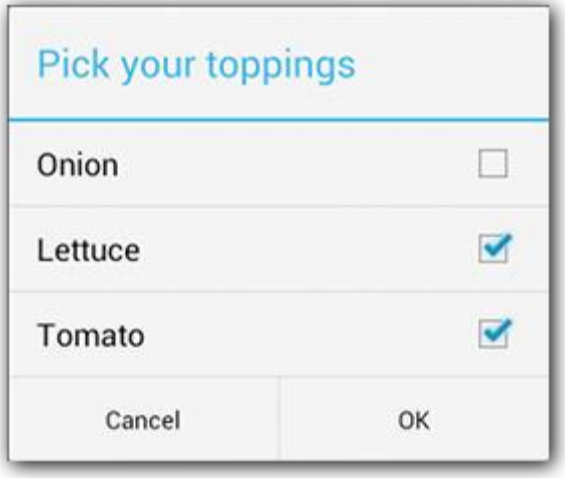
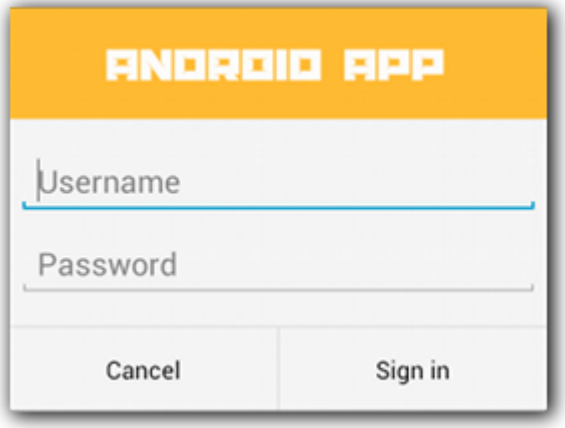


Figure 2. A list of multiple-choice items.

Custom Dialogs

Use AlertDialog and a LayoutInflator.



Custom layout file

Standard AlertDialog functionality
(setPositiveButton and setNegativeButton)

Toasts

Toasts provide lightweight feedback about an operation in a small popup.
Automatically disappear after a timeout.

Dismissing a Dialog

- When the user touches any of the action buttons created with an AlertDialog.Builder, the system dismisses the dialog for you.

- The system also dismisses the dialog when the user touches an item in a dialog list, except when the list uses radio buttons or checkboxes. Otherwise, you can manually dismiss your dialog by calling dismiss() on your DialogFragment.

- In case you need to perform certain actions when the dialog goes away, you can implement the onDismiss() method in your DialogFragment.

- You can also cancel a dialog. This is a special event that indicates the user explicitly left the dialog without completing the task. cancel()

References

https://developer.android.com/guide/topics/ui/dialogs.html