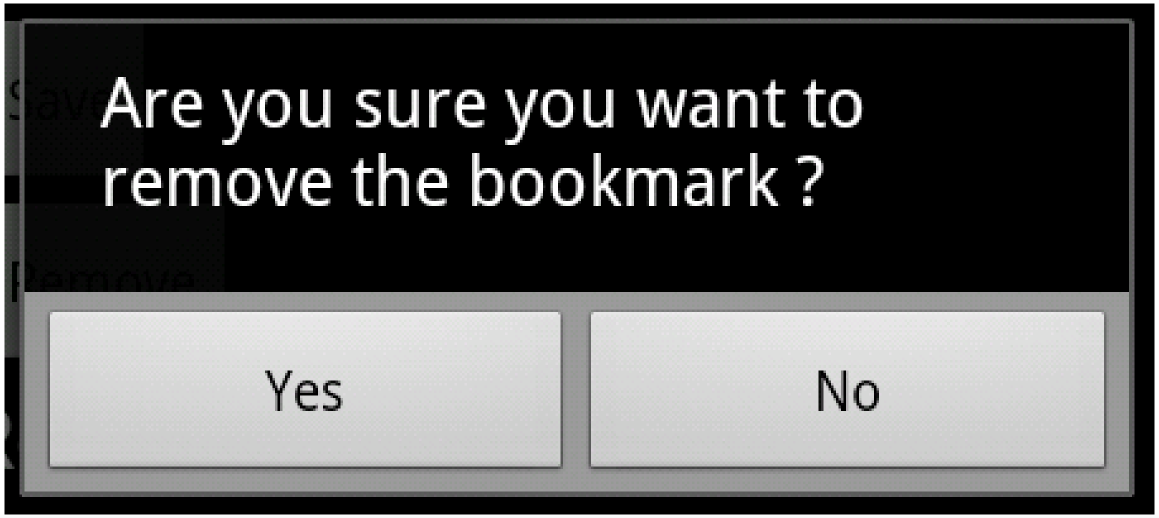
Dialogs

- A **Dialog** is a small window that appears in front of the current Activity.
- The underlying Activity loses focus and the dialog accepts all user interaction.
- Dialogs are used for notifications that should interrupt the user and to perform short tasks that directly relate to the application in progress (such as a progress bar or a login prompt).!

- The **Dialog** class is the base class for creating dialogs. You can also use one of the following subclasses:
  - AlertDialog
  - ProgressDialog
  - DatePickerDialog
  - TimePickerDialog
- If you would like to customize your own dialog, you can extend the base Dialog object or any of the subclasses listed above and define a new layout.

Alert Dialog

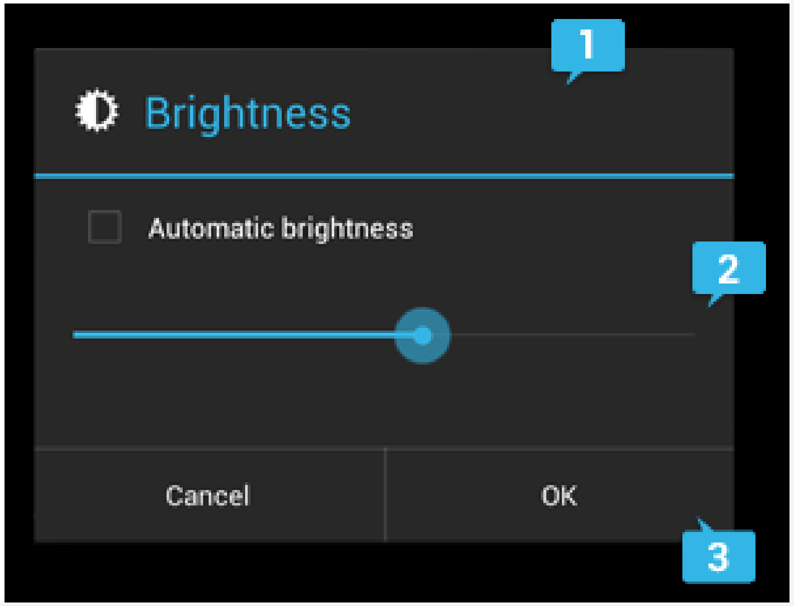
- A dialog that can manage zero, one, two, or three buttons, and/or a list of selectable items that can include checkboxes or radio buttons.
- The AlertDialog is capable of constructing most dialog user interfaces and is the suggested dialog type.


Are you sure you want to remove the bookmark ?

Alert Dialog

There are three regions of an alert dialog:

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 not be more than three action buttons in a dialog.



Alert Dialog

- The **AlertDialog.Builder** class provides APIs that allow you to create an AlertDialog with any kind of content, including a custom layout.
- To add action buttons, call the **setPositiveButton()** and **setNegativeButton()** methods.
- The set...Button() methods require a title for the button (supplied by a string resource) and a DialogInterface.OnClickListener that defines the action to take when the user presses the button.

Methods of Alert Dialog

- **setTitle(CharSequence title) –** This component is used to set the title of the alert dialog. It is optional component.
- **setMessage(CharSequence message)** – This component displays the required message in the alert dialog.
- **setCancelable(boolean cancelable) –** This component has boolean value i.e true/false. If set to false it allows to cancel the dialog box by clicking on area outside the dialog else it allows.

Alert Dialog – Action Buttons

- There are three different action buttons you can add:

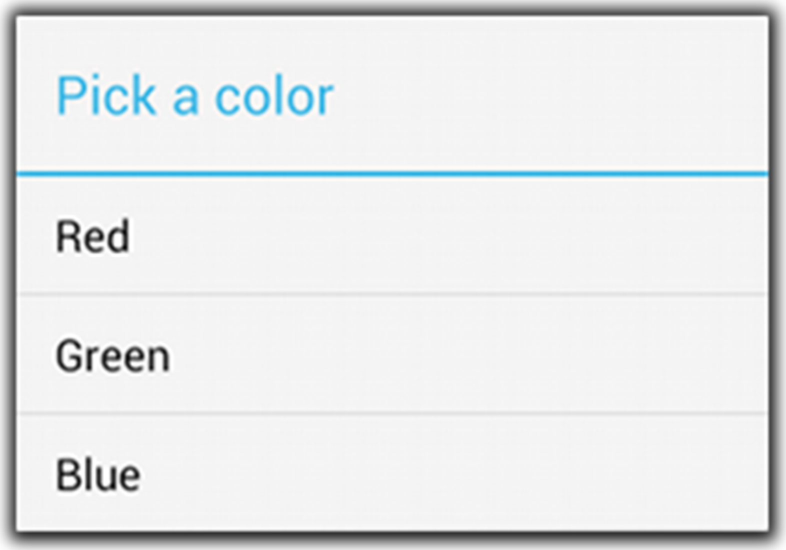
1) **Positive** : You should use this to accept and continue with the action (the "OK" action).
2) **Negative** : You should use this to cancel the action.
3) **Neutral** : You should use this when the user may not want to proceed with the action, but doesn't necessarily want to cancel. It appears between the positive and negative buttons. For example, the action might be "Remind me later."

Alert Dialog

```
AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setMessage("Are you sure you want to exit?")
.setCancelable(false)
.setPositiveButton("Yes", new DialogInterface.OnClickListener()
{
    public void onClick(DialogInterface dialog, int id) {
    MainActivity.this.finish();
}
})
.setNegativeButton("No", new DialogInterface.OnClickListener()
{
     public void onClick(DialogInterface dialog, int id) {
     dialog.cancel();
}
});
AlertDialog alert = builder.create();
alert.show();
```

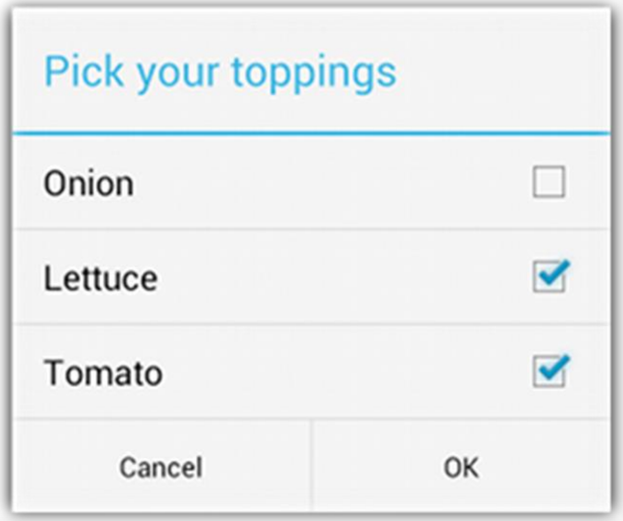
It is also possible to create an AlertDialog with a list of selectable items using the method **setItems().**

```
final String[] items = {"Red", "Green", "Blue"};
AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setTitle("Pick a color");
builder.setItems(items, new DialogInterface.OnClickListener() {
public void onClick(DialogInterface dialog, int item) {
...
}
});
AlertDialog alert = builder.create();
alert.show();
```



Using the **setMultiChoiceItems()** and **setSingleChoiceItems()** methods, it is possible to create a list of multiple-choice items (checkboxes) or single-choice items (radio buttons) inside the dialog respectively.

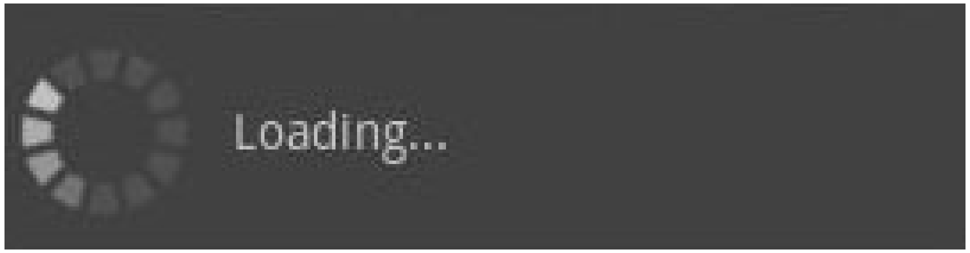
```
final String[] items = {“Onion", “Lettuce", “Tomato"};
AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setTitle("Pick your toppings");
builder.setMultiChoiceItems(items, -1, new
DialogInterface.OnMultiChoiceClickListener() {
public void onClick(DialogInterface dialog, int item)
{
}
});
AlertDialog alert = builder.create();
alert.show();
```



Progress Dialog

- A **ProgressDialog** is an extension of the AlertDialog class that can display a progress animation in the form of a spinning wheel, for a task with progress that's undefined, or a progress bar, for a task that has a defined progression.

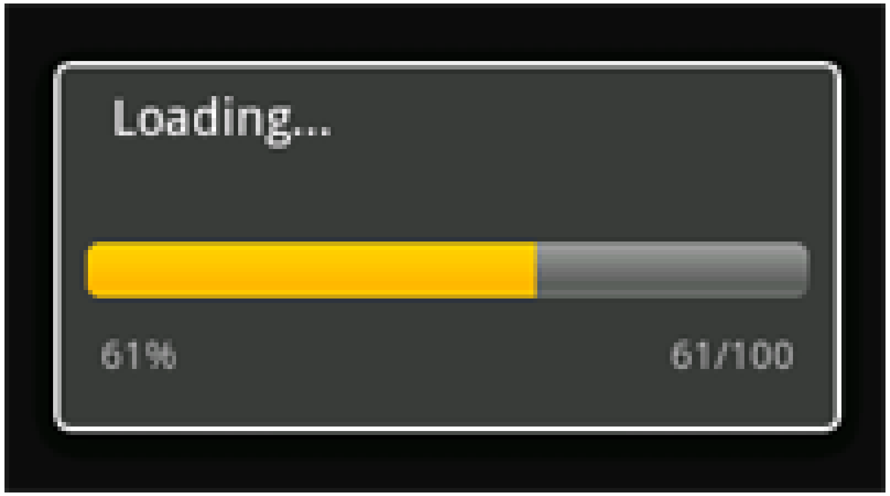
```
ProgressDialog dialog = ProgressDialog.show(MyActivity.this,
                                "", "Loading. Please wait...", true);
```



If you want to create a progress bar that shows the loading progress with granularity you can

- You can increment the amount of progress displayed in the bar by calling either setProgress(int) with a value for the total percentage completed so far or incrementProgressBy(int) with an incremental value to add to the total percentage completed so far.
- You can also specify the maximum value in specific cases where the percentage view is not useful.

```
ProgressDialog progressBar = new ProgressDialog(this);
progressBar.setCancelable(true);
progressBar.setMessage("File downloading ...");
progressBar.setProgressStyle(ProgressDialog.STYLE_H
ORIZONTAL);
progressBar.setProgress(0); //initially progress is 0
progressBar.setMax(100); //sets the maximum value 100
progressBar.show(); //displays the progress bar
```


Loading...

61%

61/100

Progress Bar

- Progress Dialog is deprecated because it prevents users from interacting with the app while progress is being displayed.
- If you need to indicate loading or indeterminate progress, you should use a **ProgressBar** in your layout, instead of using ProgressDialog.
- Progress bars are used to show progress of a task. For example, when you are uploading or downloading something from the internet, it is better to show the progress of download/upload to the user.

Progress Bar

- Progress bar supports two modes to represent progress: determinate and indeterminate.
- Use **indeterminate** mode for the progress bar when you do not know how long an operation will take. Indeterminate mode is the default for progress bar and shows a cyclic animation without a specific amount of progress indicated.

```
<ProgressBar
  android:id="@+id/indeterminateBar"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content“ />
```

Progress Bar

- Use **determinate** mode for the progress bar when you want to show that a specific quantity of progress has occurred. For example, the percent remaining of a file being retrieved, the amount records in a batch written to database, or the percent remaining of an audio file that is playing.

```
<ProgressBar
    android:id="@+id/determinateBar"
    style="@android:style/Widget.ProgressBar.Horizontal"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:progress="25“ />
```

Date Picker / Time Picker

- A dialog with a pre-defined UI that allows the user to select a date or time.
- Android **DatePicker** is a widget to select date. It allows you to select date by day, month and year. Like DatePicker, android also provides **TimePicker** to select time. Android Time Picker allows you to select the time of day in either 24 hour or AM/PM mode. The time consists of hours, minutes and clock format.
- Android provides controls for the user to pick a time or pick a date as ready-to-use dialogs.

Date Picker Dialog

- To display a DatePickerDialog using DialogFragment, you need to define a fragment class that extends DialogFragment and return a DatePickerDialog from the fragment's onCreateDialog() method.
- To define a DialogFragment for a DatePickerDialog, you must:
- Define the onCreateDialog() method to return an instance of DatePickerDialog
- Implement the DatePickerDialog.OnDateSetListener interface to receive a callback when the user sets the date.

Date Picker Dialog

```
public static class DatePickerFragment extends DialogFragment
                            implements DatePickerDialog.OnDateSetListener {

    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // Use the current date as the default date in the picker
        final Calendar c = Calendar.getInstance();
        int year = c.get(Calendar.YEAR);
        int month = c.get(Calendar.MONTH);
        int day = c.get(Calendar.DAY_OF_MONTH);

        // Create a new instance of DatePickerDialog and return it
        return new DatePickerDialog(getActivity(), this, year, month, day);
    }

    public void onDateSet(DatePicker view, int year, int month, int day) {
        // Do something with the date chosen by the user
    }
}
```

Time Picker Dialog

- To display a TimePickerDialog using DialogFragment, you need to define a fragment class that extends DialogFragment and return a TimePickerDialog from the fragment's onCreateDialog() method.
- To define a DialogFragment for a TimePickerDialog, you must:
- Define the onCreateDialog() method to return an instance of TimePickerDialog
- Implement the TimePickerDialog.OnTimeSetListener interface to receive a callback when the user sets the time.

Time Picker Dialog

```
public static class TimePickerFragment extends DialogFragment
                            implements TimePickerDialog.OnTimeSetListener {

    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // Use the current time as the default values for the picker
        final Calendar c = Calendar.getInstance();
        int hour = c.get(Calendar.HOUR_OF_DAY);
        int minute = c.get(Calendar.MINUTE);

        // Create a new instance of TimePickerDialog and return it
        return new TimePickerDialog(getActivity(), this, hour, minute,
                DateFormat.is24HourFormat(getActivity()));
    }

    public void onTimeSet(TimePicker view, int hourOfDay, int minute) {
        // Do something with the time chosen by the user
    }
}
```

Custom Dialog

- It is also possible to customize the design of a dialog. You can create your own layout for the dialog window with layout and widget elements.
- When the Dialog has been instantiated you can set your custom layout as the dialog's content view with **setContentView(int)**, passing it the layout resource ID.
- Using the method **findViewById(int)** on the dialog object it is possible to retrieve and modify its content.

```
Context mContext = getApplicationContext();
Dialog dialog = new Dialog(mContext);
dialog.setContentView(R.layout.custom_dialog);
dialog.setTitle("Custom Dialog");
TextView text = (TextView)
dialog.findViewById(R.id.text);
text.setText("Hello, this is a custom dialog!");
ImageView image = (ImageView)
dialog.findViewById(R.id.image);
image.setImageResource(R.drawable.android);
```

Dialog Fragment

- A **DialogFragment** should be used as a container for any type of dialog. The DialogFragment class provides all the controls you need to create your dialog and manage its appearance, instead of calling methods on the Dialog object.
- Using DialogFragment to manage the dialog ensures that it correctly handles lifecycle events such as when the user presses the Back button or rotates the screen. The DialogFragment class also allows you to reuse the dialog's UI as an embeddable component in a larger UI.

Dialog Fragment

```
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) {
                       ………
                   }
               })
               .setNegativeButton(R.string.cancel, new DialogInterface.OnClickListener() {
                   public void onClick(DialogInterface dialog, int id) {
                       .........
                   }
               });
        // Create the AlertDialog object and return it
        return builder.create();
    }
}
```

Activity as a Dialog

- If you want a custom dialog, you can instead display an Activity as a dialog instead of using the Dialog APIs. Simply create an activity and set its theme to **Theme.Holo.Dialog** in the <activity> manifest element.

```
<activity android:theme="@android:style/Theme.Holo.Dialog">
```

- To show an activity as a dialog only when on large screens, apply the **Theme.Holo.DialogWhenLarge** theme to the <activity> manifest element.

```
<activity android:theme=
    "@android:style/Theme.Holo.DialogWhenLarge">
```