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:

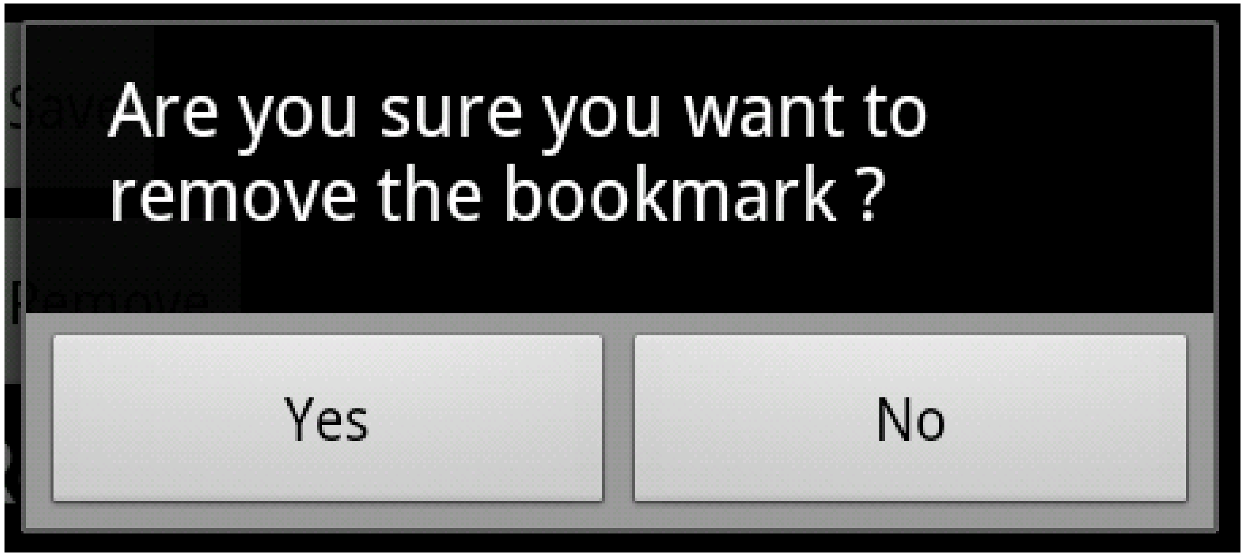
- AlertDialog
- ProgressDialog
- DatePickerDialog
- TimePickerDialog

- 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

$$\begin{tikzcd}
\text{Object} \\
\text{Dialog} \arrow[u] \\
\text{AlertDialog} \arrow[u]
\end{tikzcd}$$

- 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.

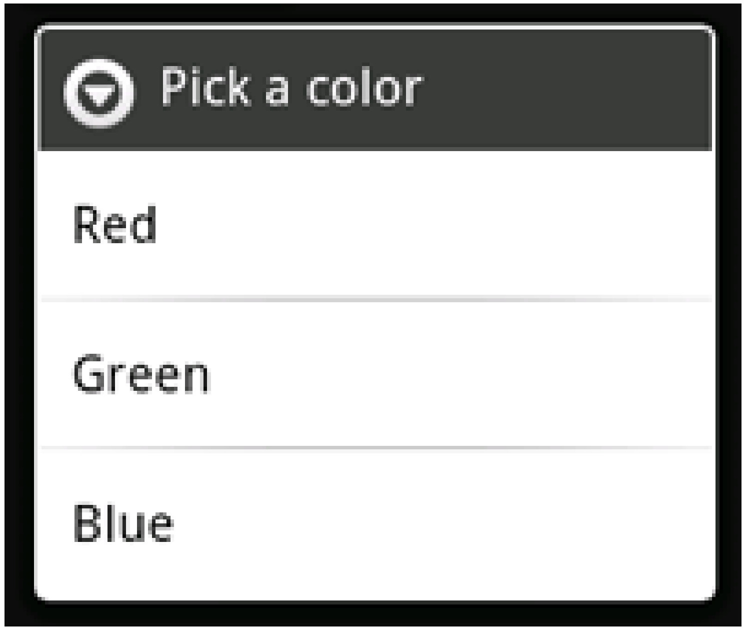


Are you sure you want to remove the bookmark ?

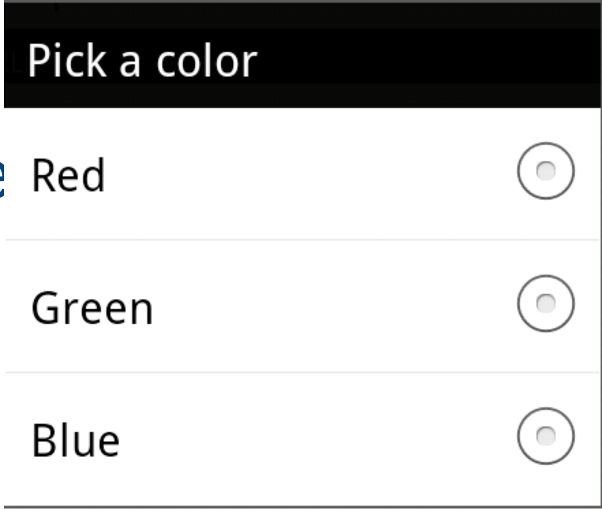
```
AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setMessage("Are you sure you want to exit?")
.setCancelable(false)
.setPositiveButton("Yes", new DialogInterface.OnClickListener()
{
     public void onClick(DialogInterface dialog, int id) {
     MainActivity.this.finish();
}
})
.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();
```



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.


Pick a color

Red ○

Green ○

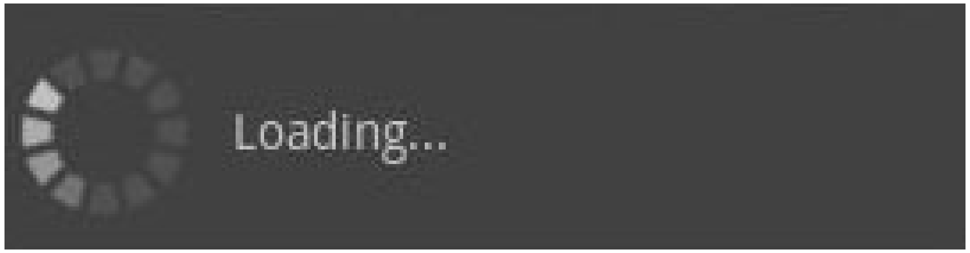
Blue ○

```
final String[] items = {"Red", "Green", Blue"};
AlertDialog.Builder builder = new
AlertDialog.Builder(this);
builder.setTitle("Pick a color");
builder.setSingleChoiceItems(items, -1, new
DialogInterface.OnClickListener() {
public void onClick(DialogInterface dialog, int item)
{
    }
    });
AlertDialog alert = builder.create();
```

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

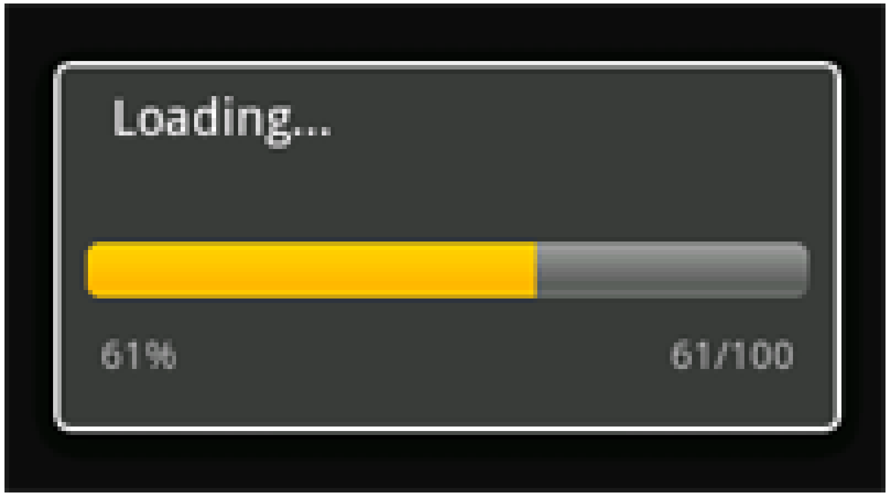
- You can increment the amount of progress displayed in the bar by calling either setProgress(int) with a value for the total percentage completed so far or incrementProgressBy(int) with an incremental value to add to the total percentage completed so far.
- You can also specify the maximum value in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

```
ProgressDialog progressDialog;

progressDialog = new ProgressDialog(mContext);

progressDialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
progressDialog.setMessage("Loading...");

progressDialog.setCancelable(false);
...
progressDialog.show();
```



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);
```

String.xml

- A string resource provides text strings for your application with optional text styling and formatting.
- A single string that can be referenced from the application or from other resource files (such as an XML layout).

A string is a simple resource that is referenced using the value provided in the `name` attribute

you can combine string resources with other simple resources in the one XML file, under one `<resources>` element.

XML file saved at `res/values/strings.xml`:`<?xml version="1.0" encoding="ut`

```
<resources>
   <string name="hello">Hello!</string>
</resources>
```

This layout XML applies a string to a View:

```
<TextView
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="@string/hello" />
```

This application code retrieves a string:

```
String string = getString(R.string.hello);
```

XML file saved at `res/values/strings.xml`:`<?xml version="1.0" encoding="ut`

```
<resources>
   <string-array name="planets_array">
      <item>Mercury</item>
      <item>Venus</item>
      <item>Earth</item>
      <item>Mars</item>
   </string-array>
</resources>
```

This application code retrieves a string array:

```
Resources res = getResources();
String[] planets = res.getStringArray(R.array.planets_array);
```

Advantages

- It centralizes the strings used by the application in a single location that is easily managed (by the developer or a non-developer).
- Strings can be defined as a resource once, and used throughout the code. Therefore, it will have consistent spelling, case and punctuation.
- Strings can be internationalized easily, allowing your application to support multiple languages with a single application package file (APK).
- Strings don’t clutter up your application code, leaving it clear and easy to maintain.