Mobile UI resources- Draw-able, Menu

Android - UI Controls

Input controls are the interactive components in your app's user interface. Android provides a wide variety of controls you can use in your UI, such as buttons, text fields, seek bars, check box, zoom buttons, toggle buttons, and many more.

A **View** is an object that draws something on the screen that the user can interact with and a **ViewGroup** is an object that holds other View (and ViewGroup) objects in order to define the layout of the user interface.

You define your layout in an XML file which offers a human-readable structure for the layout, similar to HTML. For example, a simple vertical layout with a text view and a button looks like this −

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

   <TextView android:id="@+id/text"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="I am a TextView" />

   <Button android:id="@+id/button"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="I am a Button" />
</LinearLayout>
```

Android UI Controls

There are number of UI controls provided by Android that allow you to build the graphical user interface for your app.

Sr.No.	UI Control & Description
1	TextView This control is used to display text to the user.
2	EditText EditText is a predefined subclass of TextView that includes rich editing capabilities.
3	AutoCompleteTextView The AutoCompleteTextView is a view that is similar to EditText, except that it shows a list of completion suggestions automatically while the user is typing.
4	Button A push-button that can be pressed, or clicked, by the user to perform an action.
5	ImageButton An ImageButton is an AbsoluteLayout which enables you to specify the exact location of its children. This shows a button with an image (instead of text) that can be pressed or clicked by the user.
6	CheckBox An on/off switch that can be toggled by the user. You should use check box when presenting users with a group of selectable options that are not mutually exclusive.
7	ToggleButton An on/off button with a light indicator.
8	RadioButton The RadioButton has two states: either checked or unchecked.
9	RadioGroup A RadioGroup is used to group together one or more RadioButtons.
10	ProgressBar The ProgressBar view provides visual feedback about some ongoing tasks, such as when you are performing a task in the background.
11	Spinner A drop-down list that allows users to select one value from a set.
12	TimePicker The TimePicker view enables users to select a time of the day, in either 24-hour mode or AM/PM mode.
13	DatePicker

	The DatePicker view enables users to select a date of the day.

Create UI Controls

Input controls are the interactive components in your app's user interface. Android provides a wide variety of controls you can use in your UI, such as buttons, text fields, seek bars, check box, zoom buttons, toggle buttons, and many more.

As explained in previous chapter, a view object may have a unique ID assigned to it which will identify the View uniquely within the tree. The syntax for an ID, inside an XML tag is −

```
android:id="@+id/text_id"
```

To create a UI Control/View/Widget you will have to define a view/widget in the layout file and assign it a unique ID as follows −

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

   <TextView android:id="@+id/text_id"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="I am a TextView" />
</LinearLayout>
```

Then finally create an instance of the Control object and capture it from the layout, use the following −

```
TextView myText = (TextView) findViewById(R.id.text_id);
```

Menus

Menus are a common user interface component in many types of applications. To provide a familiar and consistent user experience, you should use the Menu APIs to present user actions and other options in your activities.

Beginning with Android 3.0 (API level 11), Android-powered devices are no longer required to provide a dedicated *Menu* button. With this change, Android apps should migrate away from a dependence on the traditional 6-item menu panel and instead provide an app bar to present common user actions.

Although the design and user experience for some menu items have changed, the semantics to define a set of actions and options is still based on the Menu APIs. This guide shows how to create the three fundamental types of menus or action presentations on all versions of Android:

Options menu and app bar

The options menu is the primary collection of menu items for an activity. It's where you should place actions that have a global impact on the app, such as "Search," "Compose email," and "Settings."

See the section about Creating an Options Menu.

Context menu and contextual action mode

A context menu is a floating menu that appears when the user performs a long-click on an element. It provides actions that affect the selected content or context frame.

The contextual action mode displays action items that affect the selected content in a bar at the top of the screen and allows the user to select multiple items.

See the section about Creating Contextual Menus.

Popup menu

A popup menu displays a list of items in a vertical list that's anchored to the view that invoked the menu. It's good for providing an overflow of actions that relate to specific content or to provide options for a second part of a command. Actions in a popup menu should **not** directly affect the corresponding content—that's what contextual actions are for. Rather, the popup menu is for extended actions that relate to regions of content in your activity.

See the section about Creating a Popup Menu.

Defining a Menu in XML

For all menu types, Android provides a standard XML format to define menu items. Instead of building a menu in your activity's code, you should define a menu and all its items in an XML menu resource. You can then inflate the menu resource (load it as a Menu object) in your activity or fragment.

Using a menu resource is a good practice for a few reasons:

- It's easier to visualize the menu structure in XML.
- It separates the content for the menu from your application's behavioral code.
- It allows you to create alternative menu configurations for different platform versions, screen sizes, and other configurations by leveraging the app resources framework.

To define the menu, create an XML file inside your project's `res/menu/` directory and build the menu with the following elements:

`<menu>`
: Defines a Menu, which is a container for menu items. A `<menu>` element must be the root node for the file and can hold one or more `<item>` and `<group>` elements.

`<item>`
: Creates a MenuItem, which represents a single item in a menu. This element may contain a nested `<menu>` element in order to create a submenu.

`<group>`
: An optional, invisible container for `<item>` elements. It allows you to categorize menu items so they share properties such as active state and visibility. For more information, see the section about Creating Menu Groups.

Here's an example menu named `game_menu.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/new_game"
          android:icon="@drawable/ic_new_game"
          android:title="@string/new_game"
          android:showAsAction="ifRoom"/>
    <item android:id="@+id/help"
          android:icon="@drawable/ic_help"
          android:title="@string/help" />
</menu>
```

The `<item>` element supports several attributes you can use to define an item's appearance and behavior. The items in the above menu include the following attributes:

`android:id`
: A resource ID that's unique to the item, which allows the application to recognize the item when the user selects it.

`android:icon`
: A reference to a drawable to use as the item's icon.

`android:title`
: A reference to a string to use as the item's title.

`android:showAsAction`
: Specifies when and how this item should appear as an action item in the app bar.

These are the most important attributes you should use, but there are many more available. For information about all the supported attributes, see the Menu Resource document.

You can add a submenu to an item in any menu (except a submenu) by adding a `<menu>` element as the child of an `<item>`. Submenus are useful when your application has a lot of functions that can be organized into topics, like items in a PC application's menu bar (File, Edit, View, etc.). For example:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/file"
          android:title="@string/file" >
        <!-- "file" submenu -->
        <menu>
            <item android:id="@+id/create_new"
                  android:title="@string/create_new" />
            <item android:id="@+id/open"
                  android:title="@string/open" />
        </menu>
    </item>
</menu>
```