

# Skin Guide for Developers

## Basic Preparation

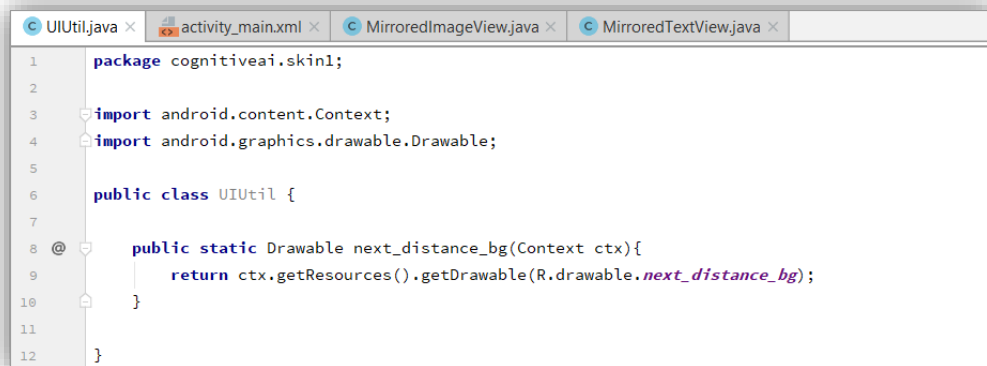
Before you start, you need to be a little bit familiar with these:

- Android developing IDE (Android Studio suggested)
- XML syntax for layout

But if you are not familiar with these, just check out some tutorial videos online and you will be good to start. Very simple.

## Skin Making Progress

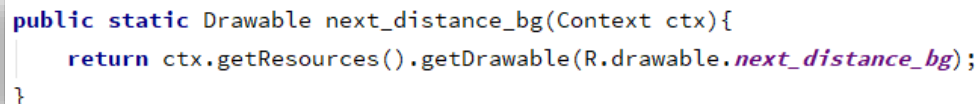
1. Start a new Android app project in your Android developing IDE. (Package name shouldn't be too long)
2. You need to prepare to write down three names for skin controller to use later:
  - Layout name (i.e. `activity_main.xml`)
  - APK file name (i.e. `app-debug.apk`)
  - Package name (i.e. `cognitiveai.skin1`)
3. Create a new java class named as “`UIUtil`” directly under the first level of your package. At the beginning of the page, the package should be the one you need to write down.



```
1 package cognitiveai.skin1;
2
3 import android.content.Context;
4 import android.graphics.drawable.Drawable;
5
6 public class UIUtil {
7
8     @
9     public static Drawable next_distance_bg(Context ctx){
10         return ctx.getResources().getDrawable(R.drawable.next_distance_bg);
11     }
12 }
```

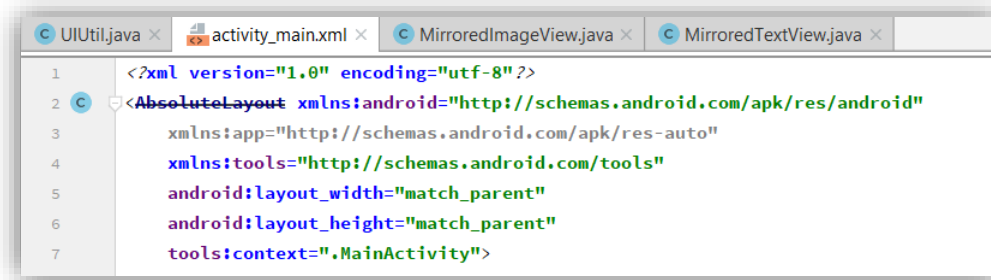
4. Create some methods for returning “`Drawable`” with the same names of the drawables for returning results in resource.

For returning the drawable resource “`next_distance_bg.png`”, the method should be named as “`next_distance_bg`”, omitting the name extension.

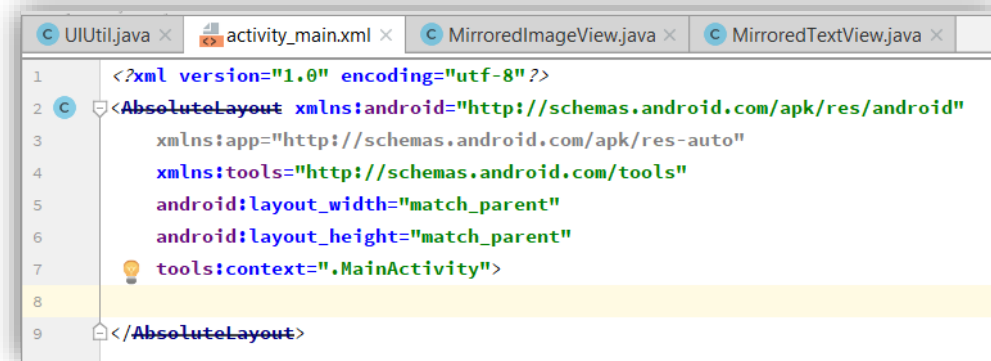


```
public static Drawable next_distance_bg(Context ctx){
    return ctx.getResources().getDrawable(R.drawable.next_distance_bg);
}
```

5. Go to any activity layout, such as “`activity_main.xml`” for an easy start.



6. In layout “Preview”, for device definition, we suggest the key definitions as below:
- Resolution: 1024 x 758: hdpi
  - API: 19
  - Target: Android 4.4
7. Set the layout theme in “Preview” to “`Black.NoTitleBar.Fullscreen`”.
8. Change the first level of the frame layout to “`AbsoluteLayout`”. (Mostly on the 2<sup>nd</sup> line and remember to change the closing to “`AbsoluteLayout`” as well.)



9. Use “`TextView`” and “`ImageView`” only (We highly recommend using “`MirroredTextView`” and “`MirroredImageView`” instead. The codes for these two customized widgets will be provided in the “Help” section.)
10. Make your own layout according to the “Key Points” as the guide.
11. After finishing building the skin, build the APK. (In Android Studio, “Build” -> “Build Bundle(s) / APK(s)” -> “Build APK(s)”). And locate the APK file (i.e. `app-debug.apk`) under the folder “`..\app\build\outputs\apk\debug`”.
12. Locate your layout file (i.e. `activity_main.xml`) under the folder “`..\app\src\main\res\layout`”.
13. Use ADB or Device File Explorer in Android Studio to push the APK file and layout file to the folder “`sdcard`”
14. Go to <https://gettingstarted.carloudy.com/> to pair your Android phone with Carloudy.
15. After you connected the Carloudy, go to Carloudy Controller app on your Android phone.
16. In the menu (Swipe right from the left edge of the screen or click the triple line icon on the top left corner of the app.), select “`Skin`”.

17. Fill the names you wrote down in step 2.
18. Click “[SEND SIGNAL](#)”, and start Google Maps navigation, you will see your result on Carloudy. Congrats!
19. If you want to stop the customized skin, click “[STOP CUSTOMIZED SKIN](#)”.

## Key Points

### Layout Guide

1. Attributes:
  - [id](#)
  - [layout\\_x](#)
  - [layout\\_y](#)
  - [layout\\_width](#)
  - [layout\\_height](#)
  - [alpha](#)
  - [gravity](#) (for TextView / MirroredTextView only)
  - [background](#) (for TextView / MirroredTextView only)
  - [text](#) (for TextView / MirroredTextView only)
  - [textColor](#) (for TextView / MirroredTextView only)
  - [textSize](#) (for TextView / MirroredTextView only)
  - [src](#) (for ImageView / MirroredImageView only)
2. One line for each attribute
3. “[/>](#)” closing must be on single line itself
4. For TextView / MirroredTextView [ids](#), you need to set certain [id](#) for each text view to function:
  - ["tv\\_distance\\_next\\_turn"](#) distance to the next turn
  - ["tv\\_next\\_street\\_name"](#) the name of the street of the next turn
  - ["tv\\_current\\_street"](#) current street name
  - ["tv\\_destination\\_name"](#) destination name
  - ["tv\\_eta"](#) estimated time of arrival
  - ["tv\\_destination\\_distance"](#) distance to the destination
  - ["tv\\_speed\\_limit"](#) speed limit
  - ["tv\\_current\\_speed"](#) current speed
  - ["tv\\_speed\\_unit"](#) speed unit
  - ["tv\\_clock"](#) clock
5. [layout\\_width](#) / [layout\\_height](#) unit: [px](#) / [dp](#)
6. [layout\\_width](#) / [layout\\_height](#) extra definition: [wrap\\_content](#) / [match\\_parent](#)
7. [layout\\_x](#) / [layout\\_y](#) unit: [px](#) / [dp](#)
8. [textSize](#) unit: [sp](#) / [dp](#)
9. [textColor](#): Hex Color code only. Format as “[#xxxxxx](#)”
10. [drawable](#) resources (including [background](#) and [src](#)) must be the same names as your methods in [UIUtil](#)
11. no spaces among the characters in each line

## Help guides

### Helper classes

#### MirroredImageView

```
public class MirroredImageView extends AppCompatImageView {  
  
    public MirroredImageView(Context context) {  
        super(context);  
    }  
  
    @Override  
    protected void onDraw(Canvas canvas) {  
        canvas.translate(getWidth(), 0);  
        canvas.scale(-1, 1);  
        super.onDraw(canvas);  
    }  
}
```

#### MirroredTextView

```
public class MirroredTextView extends AppCompatTextView {  
  
    public MirroredTextView(Context context) {  
        super(context);  
    }  
  
    @Override  
    protected void onDraw(Canvas canvas) {  
        canvas.translate(getWidth(), 0);  
        canvas.scale(-1, 1);  
        super.onDraw(canvas);  
    }  
}
```