|  |
| --- |
| Android Tutorial – Part 7 |

|  |
| --- |
| 6-29-2018 |



Table of Contents

[Introduction 2](#_Toc518042583)

[References 3](#_Toc518042584)

# Introduction

This is the part seven of the android tutorial series. It is a continuation from last week. In order to follow this successfully, it is required to have,

* A basic understanding given about android in last session.
* The environment set up.
* The project created during last tutorial, opened in Android Studio.
* AVD or an Actual device ready for app deployment.

To catch up, in the last session (Android Tutorial Part 6),

* Using Emulator to Test Sensors
* Android Motion Sensors
  + Rotational Vector Sensor
  + Gyroscope
* Android Position Sensor
  + Orientation Sensor
* Creating a Custom View in Android
  + Using paint object
  + Using canvas
* The proximity sensor
* Android environment sensors
  + Ambient Temperature
  + Light
  + Pressure
  + Relative Humidity
* Temperature

Source code for the previous tutorial:-<https://github.com/nadee158/android_tutorial_part_6.git>

With that knowledge in hand, in this session below areas will be covered,

* Android Camera API

# Android Camera API

Most Android devices have at least one camera. Some devices have a front and a back facing camera.

The Android framework includes support for various cameras and camera features available on devices, allowing developers to capture pictures and videos in the applications.

The Android framework supports capturing images and video through the **android.hardware.camera2 API** or **camera Intent**

## Things to consider before using Camera in the app

Before enabling the application to use cameras on Android devices, we should consider about how the app intends to use this hardware feature.

### Camera Requirement in app

* Is the app totally dependent on Camera – app will not work without it?
  + If yes, the camera requirement should be declared in the manifest.

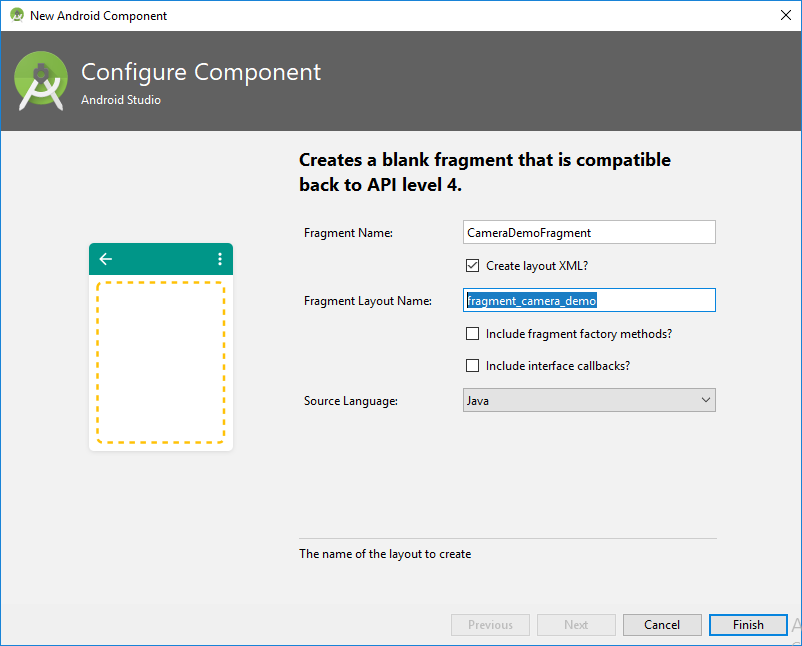
### Quick Picture or Customized Camera

* How will the application use the camera?
  + Is it for just snapping a quick picture or video clip?
    - If yes, consider Using Existing Camera Apps
  + Does the app provide a new way to use cameras?
    - If yes, build a camera using the Camera API

### Storage

The visibility, availability and security of the generated images or videos through the app

Lets now see how we could use an existing camera app to get an image to our app

1. Create a new “**Fragment**” to display the data retrieved from the Rotational Vector Sensor
   1. Right click on “**java/lk.uok.mit.fragment**”, select “**New**”🡪”**Fragment**”🡪”**Fragment (Blank)**”  
      
   2. Fill the details as shown below;  
      
      1. Fragment Name:- **CameraDemoFragment**
      2. Check create layout XML option
      3. Fragment Layout Name:- **fragment\_camera\_demo**
      4. **Uncheck both “Include fragment factory methods” and “include interface callback”**
      5. Source Language:-Java
2. Add a **menu item** to the **drawer** to access the new “**CameraDemoFragment**”
   1. Open the “**drawer\_view.xml**” inside “**res/menu**” folder and add a new menu item with id “**nav\_camera\_demo**” like below;  
      **<item  
       android:id="@+id/nav\_camera\_demo"  
       android:title="Camera Demo" />**
3. Add an entry to the switch case statement inside “**NavigationItemSelectedListener**” inside “**MainActivity**”
   1. Open “**MainActivity**” inside “**java**” folder
   2. Modify the “**setNavigationItemSelectedListene**r(” method of “**NavigationView**” inside “**onCreate**” method like shown below;  
        
      **case R.id.nav\_camera\_demo:  
       *//Open the CameraDemoFragment  
       // Begin the transaction* ft = getSupportFragmentManager().beginTransaction();  
       *// Replace the contents of the container with the new fragment* ft.replace(R.id.*fragment\_content*, new CameraDemoFragment());  
       *// Complete the changes added above* ft.commit();  
       break;**

# ConstraintLayout in Android

ConstraintLayout allows developers to create large and complex layouts with a **flat view hierarchy** (no nested view groups).

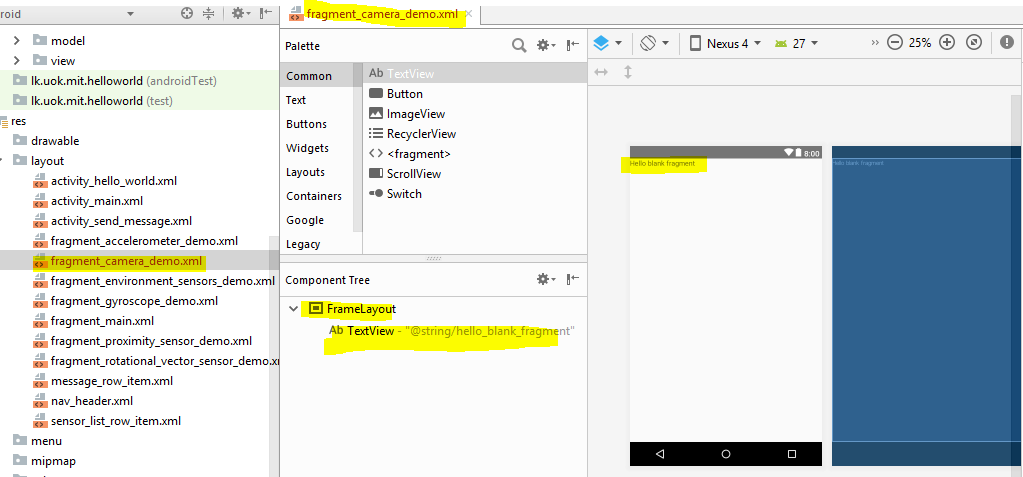
* It's **similar to RelativeLayout** in that all views are laid out according to relationships between sibling views and the parent layout,
* But it's **more flexible than RelativeLayout** and **easier to use with Android Studio's Layout Editor**.
  + All the power of ConstraintLayout is available directly from the Layout Editor's visual tools
  + The layout API and the Layout Editor were specially built for each other.
  + It is possible to build the layout with ConstraintLayout entirely by drag-and-dropping instead of editing the XML.

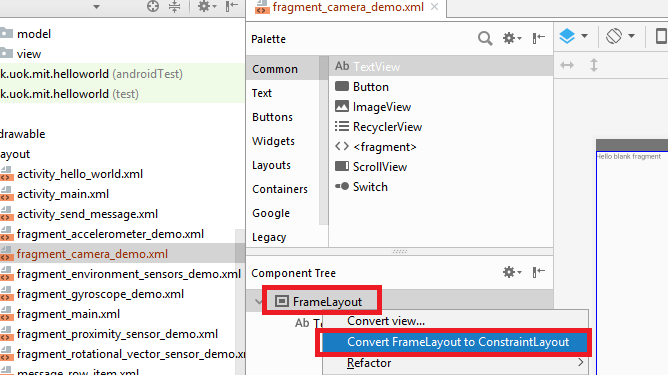
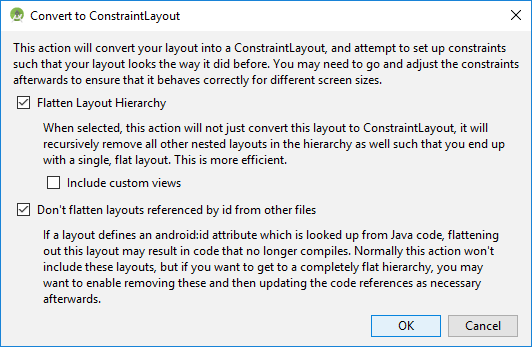
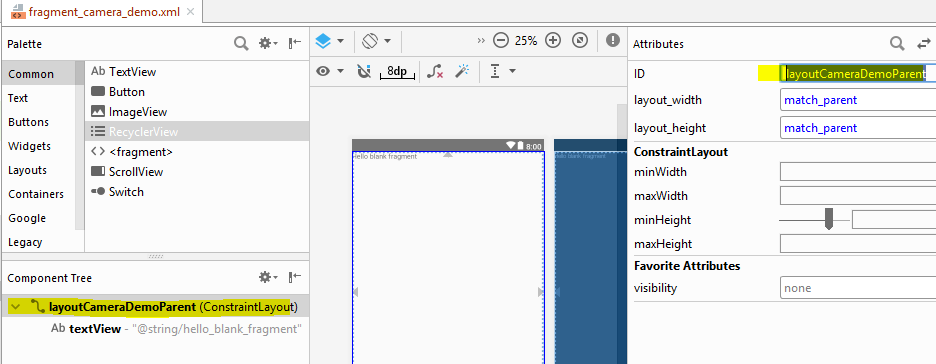
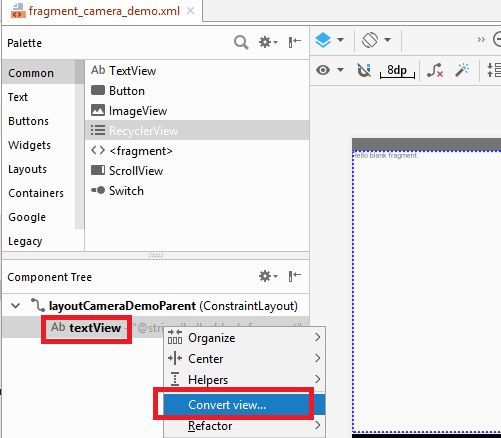
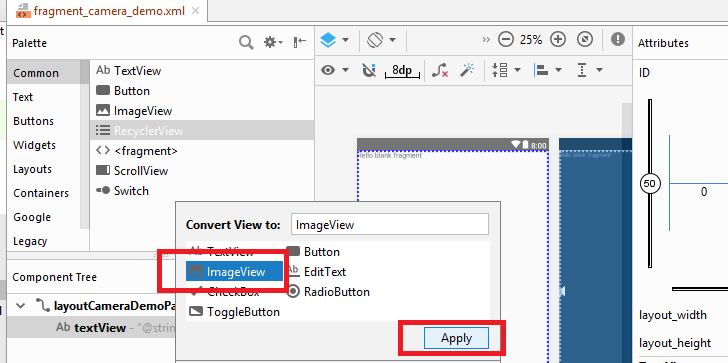
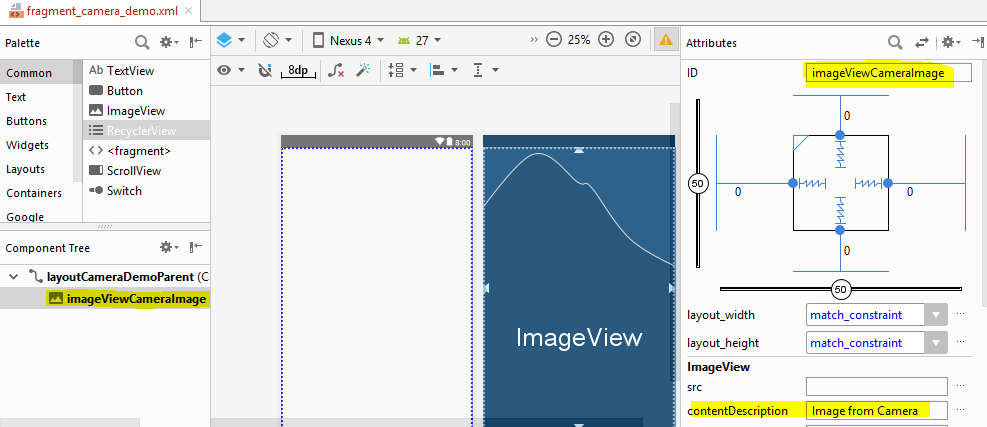
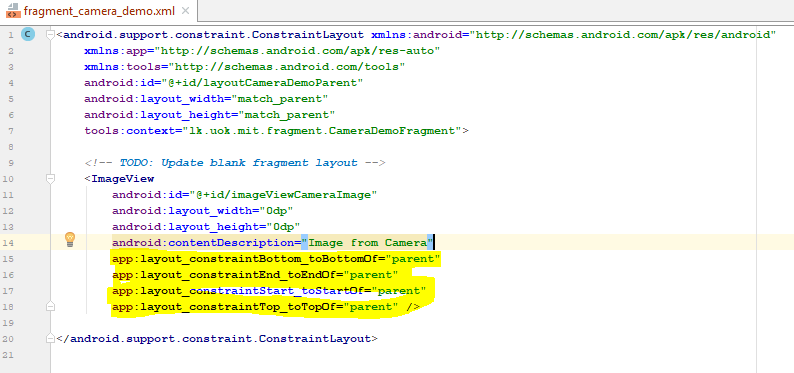
## Constraints overview

* To define a view's position in ConstraintLayout, you must add at least one horizontal and one vertical constraint for the view.
* Each constraint represents a connection or alignment to another view, the parent layout, or an invisible guideline.
* Each constraint defines the view's position along either the vertical or horizontal axis; so each view must have a minimum of one constraint for each axis, but often more are necessary.
* When you drop a view into the Layout Editor, it stays where you leave it even if it has no constraints.
* However, this is only to make editing easier; if a view has no constraints when you run your layout on a device, it is drawn at position [0,0] (the top-left corner).

Lets use a constraint layout in out fragment layout to see how to use a one in an app;

1. Modify the “**fragment\_camera\_demo.xml**” to display the image retrieved from the Camera Intent,

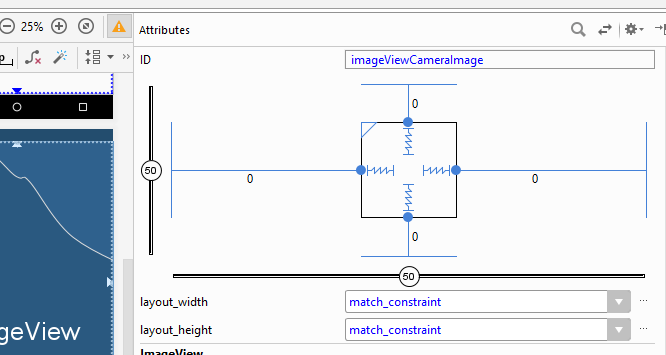
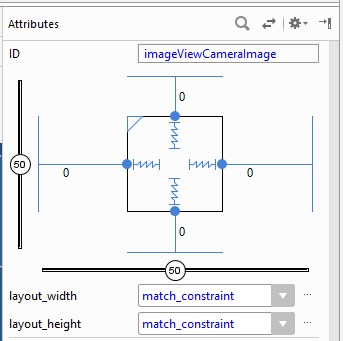
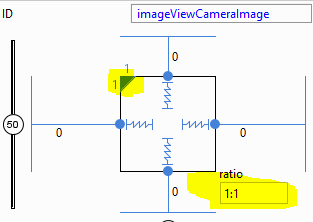
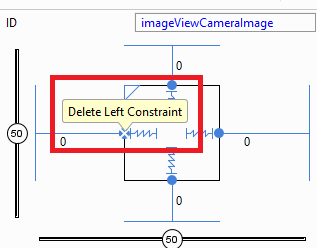
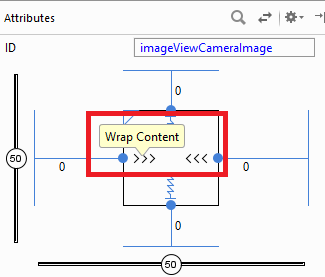
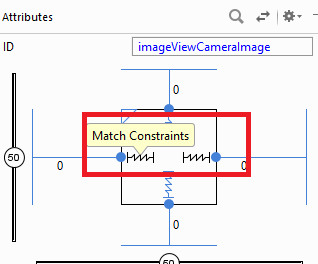
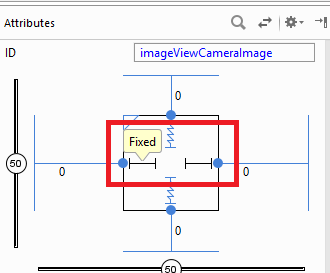
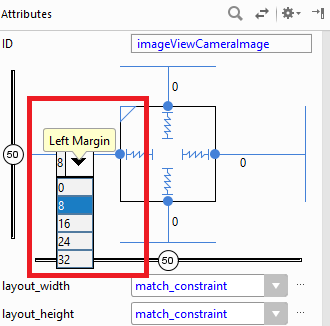
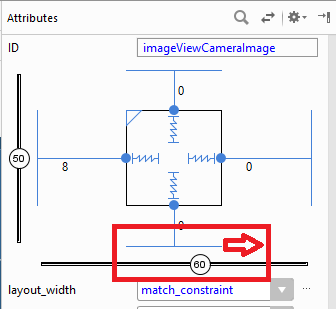
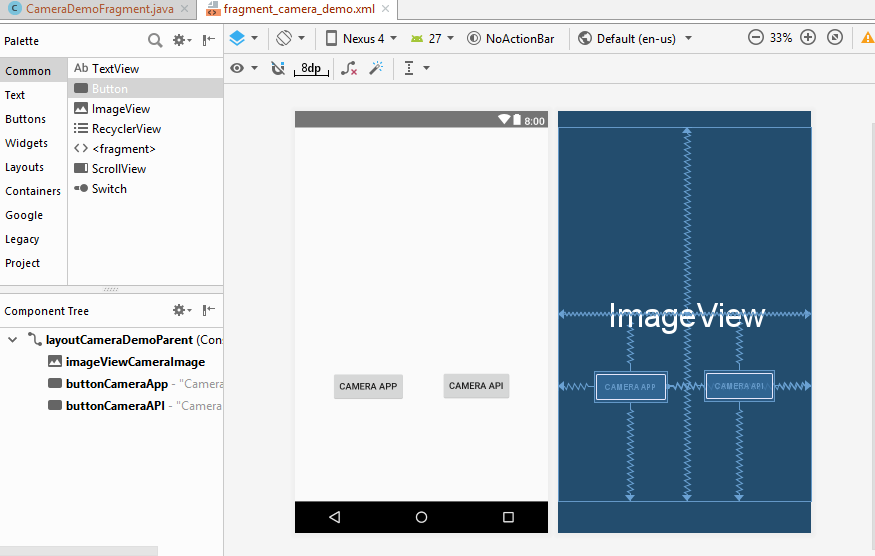
Open the **fragment\_camera\_demo.xml**  file inside “**res/layout**” folder and go to its design view, it should look like below now;  


1. Right click on the “**FrameLayout**” inside the “**Component Tree**” and select “**Convert FrameLayout to ConstraintLayout**”  
     
   1. Then the below confirm box will appear, check both checkboxes and click ok  
      
2. Change the id of the ConstraintLayout to “**layoutCameraDemoParent**” and finally, the component will look like below;  
   
3. Now to convert the “textView” to an ImageView to display the image captured from camera,
   1. Right click on “**textView**” in inside the “**Component Tree**” and select “**Convert View**”  
      
   2. Select “**ImageView**” from popup and change its id to “**imageViewCameraDemo**”  
      
   3. Change the id to “**imageViewCameraImage**” and content description to “**Image from Camera**”  
      
4. Now go to the “Text” view and check, the xml code looks like below;  
   
   1. Note the highlighted code segments, as it was mentioned above, to define a view's position in ConstraintLayout, you must add
      1. Horizontal constraint  
         LEFT:-**app:layout\_constraintStart\_toStartOf="parent"**RIGHT**:-app:layout\_constraintEnd\_toEndOf="parent"**
      2. Vertical constraint  
         TOP:- **app:layout\_constraintTop\_toTopOf="parent"**  
         BOTTOM:- **app:layout\_constraintBottom\_toBottomOf="parent"**  
           
         for the view, and each constraint represents a connection or alignment to **another view**, the **parent layout**, or **an invisible guideline**, and in our case its “PARENT LAYOUT”

## The view inspector

Near the top of the Attributes window there is a special section called the view inspector.

* It is available only for views in a constraint layout
* It includes controls for several layout attributes, as shown in figure below.

1. Now go again to the “**Design**” view of the “**fragment\_camera\_demo.xml** ” and select “**imageViewCameraImage**” from “**Component Tree**” view and check the “**Attribute window”**;  
   
2. The Attributes window includes controls for
3. size ratio – This is to set the view size to a ratio such as 16:9, if at least one of the view dimensions is set to "match constraints" (0dp).
4. To enable the ratio, click here and then enter the width:height ratio in the input that appears.  
   
5. delete constraint – to remove the constrain  
   
6. height/width mode – to change the way the height and width are calculated, click the symbol to toggle between these settings
7. https://developer.android.com/studio/images/buttons/layout-width-fixed.png - **Fixed**: to specify a specific dimension in the text box below or by resizing the view in the editor
8. https://developer.android.com/studio/images/buttons/layout-width-wrap.png - **Wrap Content**: The view expands only as much as needed to fit its contents.
9. https://developer.android.com/studio/images/buttons/layout-width-match.png - **Match Constraints**: The view expands as much as possible to meet the constraints on each side (after accounting for the view's margins)  
      
   
10. margins – to control the margin for each view  
    
11. constraint bias - the view becomes centered between the two constraints with a bias of 50% by default
12. can adjust the bias by dragging the bias slider in the Attributes window or by dragging the view  
    
13. Add two more button to the layout, and the design view and text view should look like below;
    1. “Design” view of “**fragment\_camera\_demo.xml** ”  
       
    2. Text view of “**fragment\_camera\_demo.xml** ”  
       <**android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"  
        xmlns:app="http://schemas.android.com/apk/res-auto"  
        xmlns:tools="http://schemas.android.com/tools"  
        android:id="@+id/layoutCameraDemoParent"  
        android:layout\_width="match\_parent"  
        android:layout\_height="match\_parent"  
        tools:context="lk.uok.mit.fragment.CameraDemoFragment"**>  
        <**ImageView  
        android:id="@+id/imageViewCameraImage"  
        android:layout\_width="0dp"  
        android:layout\_height="wrap\_content"  
        android:layout\_marginLeft="8dp"  
        android:layout\_marginStart="8dp"  
        android:contentDescription="Image from Camera"  
        app:layout\_constraintBottom\_toBottomOf="parent"  
        app:layout\_constraintEnd\_toEndOf="parent"  
        app:layout\_constraintStart\_toStartOf="parent"  
        app:layout\_constraintTop\_toTopOf="parent"** />  
         
        <**Button  
        android:id="@+id/buttonCameraApp"  
        android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content"  
        android:text="Camera App"  
        app:layout\_constraintBottom\_toBottomOf="parent"  
        app:layout\_constraintEnd\_toEndOf="parent"  
        app:layout\_constraintEnd\_toStartOf="@+id/buttonCameraAPI"  
        app:layout\_constraintHorizontal\_bias="0.5"  
        app:layout\_constraintStart\_toStartOf="parent"  
        app:layout\_constraintTop\_toBottomOf="@+id/imageViewCameraImage"  
        app:layout\_constraintVertical\_bias="0.363"** />  
         
        <**Button  
        android:id="@+id/buttonCameraAPI"  
        android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content"  
        android:text="Camera API"  
        app:layout\_constraintBottom\_toBottomOf="parent"  
        app:layout\_constraintEnd\_toEndOf="parent"  
        app:layout\_constraintHorizontal\_bias="0.5"  
        app:layout\_constraintStart\_toEndOf="@+id/buttonCameraApp"  
        app:layout\_constraintTop\_toBottomOf="@+id/imageViewCameraImage"  
        app:layout\_constraintVertical\_bias="0.358"** />  
         
       </**android.support.constraint.ConstraintLayout**>

After getting familiar with the constrain layout and its specific view inspector, lets get back to our main objective, to display an image captured from camera through the default camera app;

1. Add permission to write the captured image file to external storage, and to access camera by adding below lines to “**AndroidManifest.xml**” file inside “manifests” folder  
   *<!-- Permission to access memory -->*<**uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE"** />  
   *<!-- Permission to access camera -->*<**uses-permission android:name="android.permission.CAMERA"** />
2. Modify the “**CameraDemoFragment**” inside java, to acquire the rotation vector sensor and get sensor data from it to determine the rotation of the device along z axis

Modify the “**onCreateView**” method inside “**CameraDemoFragment**” and add the text to display on title bar as shown below;  
@Override  
**public** View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {  
 *//set the text appear in title bar* getActivity().setTitle(**"Camera Demo"**);  
 **return** inflater.inflate(R.layout.***fragment\_camera\_demo***, container, **false**);  
}



Source code for this tutorial part can be found in Git Repository given below: - <https://github.com/nadee158/android_tutorial_part_7.git>

# References

Ableson, F. (2009, June 16). *Tapping into Android's sensors*. Retrieved from www.ibm.com: https://www.ibm.com/developerworks/library/os-android-sensor/index.html

*Android AutoCompleteTextView Example*. (2018, January 1). Retrieved from JavaTPoint: https://www.javatpoint.com/android-autocompletetextview-example

*Application Fundamentals*. (2018, May 2018). Retrieved from Android Developers: https://developer.android.com

CHUGH, A. (2018, April 2). *Android AutoCompleteTextView Example Tutorial*. Retrieved from JournalDev: https://www.journaldev.com/9574/android-autocompletetextview-example-tutorial

*Creating and Using Fragments*. (2015, 01 01). Retrieved from guides.codepath.com: https://guides.codepath.com/android/creating-and-using-fragments

google-developer-training. (2018, June 18). *Working with sensor data*. Retrieved from google-developer-training.gitbooks.io: https://google-developer-training.gitbooks.io/android-developer-advanced-course-practicals/unit-1-expand-the-user-experience/lesson-3-sensors/3-1-p-working-with-sensor-data/3-1-p-working-with-sensor-data.html

Green, P. (2015, April 03). *Using Android Sensors in Your App*. Retrieved from www.sitepoint.com: https://www.sitepoint.com/using-android-sensors-application/

Hathibelagal, A. (2017, January 27). *Android Sensors in Depth: Proximity and Gyroscope*. Retrieved from Envato.com: https://code.tutsplus.com/tutorials/android-sensors-in-depth-proximity-and-gyroscope--cms-28084

*Sensors Overview*. (2018, June 19). Retrieved from Android Developers: https://developer.android.com/guide/topics/sensors/sensors\_overview

Tutorialspoint. (2018, June 18). *Android - Sensors*. Retrieved from Tutorialspoint: https://www.tutorialspoint.com/android/android\_sensors.htm