Android drag and drop advanced concepts

Android drag/drop process

In android, the Drag and Drop process contains 4 steps or states:

- **Started**
- **Continuing**
- **Dropped**
- **Ended**

Started

In response to the user's gesture **(for example, on a long press)** to begin a drag, your application calls **startDrag()** to tell the system to start a drag.

The **startDrag()** method arguments will provide data to be dragged, metadata for this data and a call-back for drawing the drag shadow.

After that, the system sends a drag event with action type **ACTION_DRAG_STARTED** to the drag event listeners for all the View objects in the current layout.

Continuing

The user continues the drag, and the system now sends **ACTION_DRAG_ENTERED** action followed by **ACTION_DRAG_LOCATION** action to the registered drag event listener for the View. The listener may choose to alter its View object's appearance in response to the event or can react by highlighting its View.

The drag event listener receives a **ACTION_DRAG_EXITED** action after the user has moved the drag shadow outside the bounding box of the View.

Dropped

User drops the dragged item, and the system sends the **ACTION_DROP** event only if the user drops the drag shadow within the bounding box of a View which listener is registered to receive drag events.

For example: If there are two views **View1** and **View2** and **View1** is registered for dragListener but **view2** is not, then if user drops the item in **View1** then the drop action will be called for **View1** but if user drops the item in **view2** then no drop action will be called for **View2**.

Ended

The drag-and-drop operation has ended, and the system sends out a drag event with action type **ACTION_DRAG_ENDED**. All of the views that were listening for the drag-and-drop events now get the **ACTION_DRAG_ENDED** event.

OnDragListener

To receive the drag-and-drop events, we need to register a **View.OnDragListener** on the View using **setOnDragListener()**.

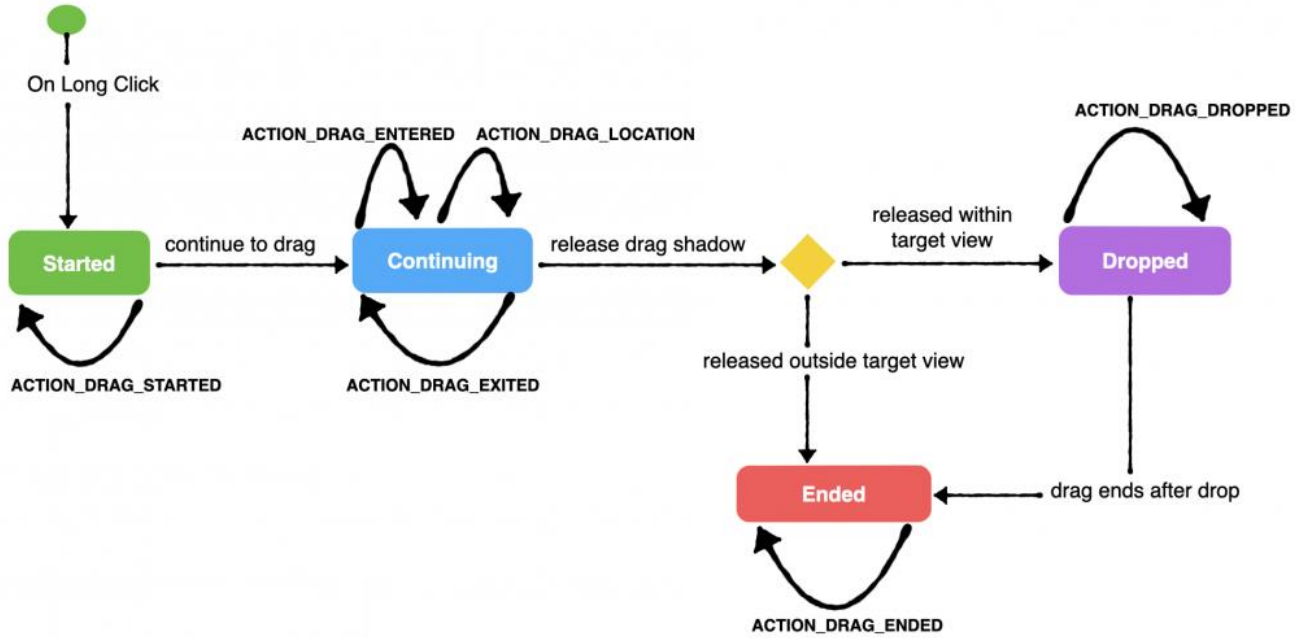
The **View.OnDragListener** having a callback method, **onDrag(View v, DragEvent event).**

- **v**: the **view** that received the event.
- **event**: **DragEvent** object, which contains all the information about the drag-and-drop event, including the location of the event, the drag action (state), and the data it carries.

Drag events

When **startDrag()** function is called, then the system identifies the type of drag event that is currently taking place and return the drag event to the application. It sends out a drag event in the form of a **DragEvent** object.

getAction() is used to identify which type of drag event is taking place.



Sr.No.	Constants & Description
1	**ACTION_DRAG_STARTED** Signals the start of a drag and drop operation.
2	**ACTION_DRAG_ENTERED** Signals to a View that the drag point has entered the bounding box of the View.
3	**ACTION_DRAG_LOCATION** Sent to a View after ACTION_DRAG_ENTERED if the drag shadow is still within the View object's bounding box.
4	**ACTION_DRAG_EXITED** Signals that the user has moved the drag shadow outside the bounding box of the View.
5	**ACTION_DROP** Signals to a View that the user has released the drag shadow, and the drag point is within the bounding box of the View.
6	**ACTION_DRAG_ENDED** Signals to a View that the drag and drop operation has concluded.

Following are few important and most frequently used methods available as a part of DragEvent class.

Sr.No.	Constants & Description
1	**int getAction()** Inspect the action value of this event..
2	**ClipData getClipData()** Returns the ClipData object sent to the system as part of the call to startDrag().
3	**ClipDescription getClipDescription()**

	Returns the ClipDescription object contained in the ClipData.
4	**boolean getResult()** Returns an indication of the result of the drag and drop operation.
5	**float getX()** Gets the X coordinate of the drag point.
6	**float getY()** Gets the Y coordinate of the drag point.
7	**String toString()** Returns a string representation of this DragEvent object.

ClipData is **a complex type containing one or more Item instances, each of which can hold one or more representations of an item of data**. For display to the user, it also has a label. A ClipData contains a ClipDescription , which describes important meta-data about the clip

```
package com.example.dragdropexample;

import android.content.ClipData;
import android.content.ClipDescription;
import android.graphics.Color;
import android.graphics.PorterDuff;
import android.os.Bundle;
import android.util.Log;
import android.view.DragEvent;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.LinearLayout;
import android.widget.TextView;
import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;
```

```
public class MainActivity extends Activity {
  ImageView img;
  String msg;
  private android.widget.RelativeLayout.LayoutParams layoutParams;

  @Override
```

```
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  img=(ImageView)findViewById(R.id.imageView);

  img.setOnLongClickListener(new View.OnLongClickListener() {
    @Override
    public boolean onLongClick(View v) {
      ClipData.Item item = new ClipData.Item((CharSequence)v.getTag());
      String[] mimeTypes = {ClipDescription.MIMETYPE_TEXT_PLAIN};

      ClipData dragData = new ClipData(v.getTag().toString(),mimeTypes, item);
      View.DragShadowBuilder myShadow = new View.DragShadowBuilder(img);

      v.startDrag(dragData,myShadow,null,0);
      return true;
    }
  });

  img.setOnDragListener(new View.OnDragListener() {
    @Override
    public boolean onDrag(View v, DragEvent event) {
      switch(event.getAction()) {
        case DragEvent.ACTION_DRAG_STARTED:
        layoutParams = (RelativeLayout.LayoutParams)v.getLayoutParams();
        Log.d(msg, "Action is DragEvent.ACTION_DRAG_STARTED");

        // Do nothing
        break;

        case DragEvent.ACTION_DRAG_ENTERED:
        Log.d(msg, "Action is DragEvent.ACTION_DRAG_ENTERED");
        int x_cord = (int) event.getX();
        int y_cord = (int) event.getY();
        break;

        case DragEvent.ACTION_DRAG_EXITED :
        Log.d(msg, "Action is DragEvent.ACTION_DRAG_EXITED");
        x_cord = (int) event.getX();
        y_cord = (int) event.getY();
        layoutParams.leftMargin = x_cord;
        layoutParams.topMargin = y_cord;
        v.setLayoutParams(layoutParams);
        break;

        case DragEvent.ACTION_DRAG_LOCATION  :
        Log.d(msg, "Action is DragEvent.ACTION_DRAG_LOCATION");
        x_cord = (int) event.getX();
        y_cord = (int) event.getY();
        break;
```

```
        case DragEvent.ACTION_DRAG_ENDED   :
        Log.d(msg, "Action is DragEvent.ACTION_DRAG_ENDED");

        // Do nothing
        break;

        case DragEvent.ACTION_DROP:
        Log.d(msg, "ACTION_DROP event");

        // Do nothing
        break;
        default: break;
      }
      return true;
    }
  });

  img.setOnTouchListener(new View.OnTouchListener() {
    @Override
    public boolean onTouch(View v, MotionEvent event) {
      if (event.getAction() == MotionEvent.ACTION_DOWN) {
        ClipData data = ClipData.newPlainText("", "");
        View.DragShadowBuilder shadowBuilder = new View.DragShadowBuilder(img);

        img.startDrag(data, shadowBuilder, img, 0);
        img.setVisibility(View.INVISIBLE);
        return true;
      } else {
        return false;
      }
    }
  });
 }
}
```

Following will be the content of **res/layout/activity_main.xml** file −

In the following code abc indicates the logo of tutorialspoint.com

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

    <LinearLayout
        android:id="@+id/top_layout"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:background="@color/colorPrimary"
```

```
        android:gravity="center"
        android:orientation="vertical">

        <Button
            android:id="@+id/btn_draggable"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/btn_text" />

        <TextView
            android:id="@+id/tv_draggable"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:gravity="center"
            android:layout_marginTop="@dimen/margin_10dp"
            android:text="@string/tv_text"
            android:textColor="@android:color/white"
            android:textSize="20sp"
            android:textStyle="bold" />
    </LinearLayout>

    <LinearLayout
        android:id="@+id/bottom_layout"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:gravity="center"
        android:background="@color/colorAccent"
        android:orientation="vertical" />

</LinearLayout>
```

Conclusion

The Android drag-and-drop framework provides you with a flexible yet powerful API for dragging data across your app. It allows you to handle all of the stages in the drag-and-drop process in a very convenient way — both in terms of data and the UI.

Your app might need a custom solution, but in most of the drag-and-drop implementations, this API will be sufficient; you might just have to figure out the best way to set it up for your needs.

****