拖放

内容快览

- 允许用户使用图形化手势在您的活动布局中移动数据。
- 支持数据移动以外的操作。
- 仅可在单一应用中使用。
- 需要 API 11。

本文内容

关键类

- `View`
- `OnLongClickListener`
- `OnDragListener`
- `DragEvent`
- `DragShadowBuilder`
- `ClipData`
- `ClipDescription`

相关示例

- Honeycomb Gallery。
- Api 演示中的 DragAndDropDemo.java 和 DraggableDot.java。

另请参阅

- 内容提供程序
- 输入事件

借助 Android 拖放框架，可允许您的用户使用图形化拖放手势，将数据从当前布局中的一个视图移到另一个视图。 该框架包括拖拽事件类、拖拽侦听器以及帮助程序方法和类。

尽管该框架主要为数据移动而设计，但也可以将其用于其他 UI 操作。 例如，您可以创建一个应用，在用户将一个颜色图标拖到另一个图标上面时进行颜色混合。 不过，本主题的其余部分将从数据移动方面介绍该框架。

概览

当用户做出某种您识别为开始拖拽数据的手势时，拖放操作开始。 作为响应，您的应用会告知系统正在开始拖拽。 系统回调应用，以获取正在拖拽的数据的表示。 用户手指在当前布局上移动此表示（“拖拽阴影”）的过程中，系统将拖拽事件发送到与布局中的 `View` 对象相关联的拖拽事件侦听器对象和拖拽事件回调方法。 用户释放拖拽阴影后，系统立即结束拖拽操作。

从实现 `View.OnDragListener` 的类创建拖拽事件侦听器对象（“listeners”）。 使用视图对象的 `setOnDragListener()` 方法为视图设置拖拽事件侦听器对象。 每个视图对象还有一个 `onDragEvent()` 回调方法。 在拖拽事件侦听器和回调方法部分对以上两者进行了更详细的说明。

> **注：**为简便起见，以下部分将接收拖拽事件的例程称为“拖拽事件侦听器”，即便它实际可能是一个回调方法也是如此。

在开始拖拽时，将您想要移动的数据和描述此数据的元数据均包含在系统调用中。 在拖拽期间，系统会将拖拽事件发送到布局中每个视图的拖拽事件侦听器或回调方法。 这些侦听器或回调方法可使用元数据来确定其在数据被放下时是否想要接受这些数据。 如果用户将数据放到某个视图对象上，并且该视图对象的侦听器或回调方法之前已告知系统它想要接受放下的数据，则系统会将该数据发送到拖拽事件中的侦听器或回调方法。

您的应用通过调用 `startDrag()` 方法告知系统开始拖拽。 这将告知系统开始发送拖拽事件。该方法还会发送正在拖拽的数据。

您可以为当前布局中任意已连接的视图调用 `startDrag()`。 系统仅使用视图对象获取布局中的全局设置访问权限。

应用调用 `startDrag()` 之后，该过程的剩余部分将使用系统发送给当前布局中的视图对象的事件。

拖放过程

拖放过程基本包含四个步骤或状态：

开始

为响应用户开始拖拽的手势，您的应用将调用 `startDrag()`，告知系统开始拖拽。 参数 `startDrag()` 提供了将要拖拽的数据、此数据的元数据，以及用于绘制拖拽阴影的回调。

系统首先通过回调应用进行响应，以获取拖拽阴影。然后在设备上显示拖拽阴影。

接下来，系统将具有操作类型 `ACTION_DRAG_STARTED` 的拖拽事件发送到当前布局中所有视图对象的拖拽事件侦听器。 要继续接收拖拽事件（包括可能的放下事件），拖拽事件侦听器必须返回 `true`。 这将在系统中注册该侦听器。只有已注册的侦听器才能继续接收拖拽事件。 此时，侦听器也可以更改其视图对象的外观，以表明该侦听器可以接受放下事件。

如果拖拽事件侦听器返回 `false`，则它将不会接收当前操作的拖拽事件，直至系统发送具有操作类型 `ACTION_DRAG_ENDED` 的拖拽事件为止。 通过发送 `false`，侦听器告知系统，它对拖拽操作不感兴趣，不想接受拖拽的数据。

继续

用户继续拖拽。在拖拽阴影与某个视图对象的边界框相交时，系统将向视图对象的拖拽事件侦听器（如果该侦听器已注册接收事件）发送一个或多个拖拽事件。 侦听器可以选择更改其视图对象的外观以响应该事件。 例如，如果该事件指示拖拽阴影已进入视图的边界框（操作类型 `ACTION_DRAG_ENTERED`），则侦听器可通过突出显示其视图来做出反应。

放下

用户在可接受数据的视图的边界框内释放拖拽阴影。 系统向视图对象的侦听器发送具有操作类型 `ACTION_DROP` 的拖拽事件。该拖拽事件包含在启动操作的 `startDrag()` 调用中传递给系统的数据。 如果成功执行了用于接受放下事件的代码，侦听器预期将向系统返回布尔值 `true`。

请注意，仅当用户在其侦听器已注册接收拖拽事件的视图的边界框内放下拖拽阴影时，才会发生这一步。 如果用户在其他任何情况下释放拖拽阴影，将不会发送任何 `ACTION_DROP` 拖拽事件。

结束

在用户释放拖放阴影并且系统发出（如果有必要）具有操作类型 `ACTION_DROP` 的拖拽事件后，系统将发出具有操作类型 `ACTION_DRAG_ENDED` 的拖拽事件，指示拖拽操作结束。 不论用户在哪里释放拖放阴影都会执行此操作。 该事件将被发送到已注册接收拖拽事件的每个侦听器（即使该侦听器接收过 `ACTION_DROP` 事件）。

设计拖放操作部分对以上四个步骤分别进行了更详尽的说明。

拖拽事件侦听器和回调方法

拖拽事件侦听器和回调方法

视图使用实现 `View.OnDragListener` 的拖拽事件侦听器，或者使用其 `onDragEvent(DragEvent)` 回调函数来接收拖拽事件。 当系统调用该方法或侦听器时，会向其传递给一个 `DragEvent` 对象。

在多数情况下，您可能希望使用侦听器。在设计 UI 时，通常不会将视图类划入子类，但使用回调方法会迫使您这样做，以便重写该方法。 相比之下，您可以实现一个侦听器类，然后将其与多个不同的视图对象配合使用。 也可以将其实现为匿名内联类。要设置视图对象的侦听器，请调用 `setOnDragListener()`。

您可能同时拥有视图对象的侦听器和回调方法。如果出现这种情况，系统会首先调用侦听器。 除非侦听器返回 `false`，否则系统不会调用回调方法。

`onDragEvent(DragEvent)` 方法和 `View.OnDragListener` 的组合与用于触摸事件的 `onTouchEvent()` 和 `View.OnTouchListener` 组合类似。

拖拽事件

系统以 `DragEvent` 对象的形式发出拖拽事件。该对象包含的操作类型会告知侦听器拖放过程中所发生的情况。 根据操作类型，该对象还包含其他数据。

要获取操作类型，侦听程序可调用 `getAction()`。可能的值有六个，由 `DragEvent` 类中的常量定义。 表 1 中列出了这些值。

`DragEvent` 对象还包含您的应用在 `startDrag()` 调用中提供给系统的数据。 其中一些数据仅对特定的操作类型有效。表 2 中概括了每种操作类型的有效数据。在设计拖放操作部分也详尽描述了该数据及其适用的事件。

表 1. DragEvent 操作类型

getAction() 值	含义
`ACTION_DRAG_STARTED`	视图对象的拖拽事件侦听器在应用调用 `startDrag()` 并获得拖拽阴影之后立即收到此事件操作类型。
`ACTION_DRAG_ENTERED`	视图对象的拖拽事件侦听器在拖拽阴影刚刚进入视图的边界框时收到此事件操作类型。 这是侦听器在拖拽阴影进入边界框时收到的第一个事件操作类型。 如果侦听器想要继续接收此操作的拖拽事件，必须向系统返回布尔值 `true`。
`ACTION_DRAG_LOCATION`	视图对象的拖拽事件侦听器在收到 `ACTION_DRAG_ENTERED` 事件后并且拖拽阴影仍在该视图的边界框内时收到此事件操作类型。
`ACTION_DRAG_EXITED`	视图对象的拖拽事件侦听器在收到 `ACTION_DRAG_ENTERED` 和至少一个 `ACTION_DRAG_LOCATION` 事件，并且用户已将拖拽阴影移到该视图的边界框以外后收到此事件操作类型。
`ACTION_DROP`	视图对象的拖拽事件侦听器在用户将拖拽阴影释放到视图对象上面时收到此事件操作类型。 仅当视图对象的侦听器在响应 `ACTION_DRAG_STARTED` 拖拽事件时返回了布尔值 `true`，才会将此操作类型发送至该侦听器。 如果用户将拖拽阴影释放到未注册侦听器的视图上或用户将拖拽阴影释放到不属于当前布局的任何视图上，则不会发送此操作类型。 侦听器如果成功处理了放下操作，预期将返回布尔值 `true`。 否则，它应该返回 `false`。
`ACTION_DRAG_ENDED`	视图对象的拖拽事件侦听器在系统结束拖拽操作时收到此事件操作类型。 此操作类型不一定在 `ACTION_DROP` 事件之后。 如果系统发送了 `ACTION_DROP`，收到 `ACTION_DRAG_ENDED` 操作类型并不表示放下操作成功。 侦听器必须调用 `getResult()` 以获得响应 `ACTION_DROP` 时所返回的值。 如果没有发送 `ACTION_DROP` 事件，则 `getResult()` 将返回 `false`。

表 2. 按操作类型列出的有效 DragEvent 数据

`getAction()` 值	`getClipDescription()` 值	`getLocalState()` 值	`getX()` 值	`getY()` 值	`getClipData()` 值	`getResult()` 值
`ACTION_DRAG_STARTED`	X	X	X			
`ACTION_DRAG_ENTERED`	X	X	X	X		
`ACTION_DRAG_LOCATION`	X	X	X	X		
`ACTION_DRAG_EXITED`	X	X				
`ACTION_DROP`	X	X	X	X	X	

ACTION_DRAG_ENDED	X	X				X

getAction()、describeContents()、writeToParcel() 和 toString() 方法始终返回有效数据。

如果某个方法不包含特定操作类型的有效数据，则根据其结果类型，将会返回 null 或 0。

拖拽阴影

在拖放操作期间，系统会显示用户拖拽的图像。 对于数据移动，此图像表示正在拖拽的数据。对于其他操作，此图像表示拖拽操作的某个方面。

此图像被称为拖拽阴影。您使用为 View.DragShadowBuilder 对象声明的方法创建拖拽阴影，然后在使用 startDrag()开始拖拽时将其传递给系统。 作为系统对 startDrag() 的响应的一部分，系统会调用您在 View.DragShadowBuilder 中定义的回调方法以获取拖拽阴影。

View.DragShadowBuilder 类有两个构造函数：

View.DragShadowBuilder(View)

> 此构造函数可接受应用的任何 View 对象。该构造函数在 View.DragShadowBuilder 对象中存储视图对象，因此在回调期间，您可以在构造拖拽阴影时访问它。 它不一定必须与用户选择开始拖拽操作的视图（如果有）相关联。
>
> 如果使用此构造函数，则无需扩展 View.DragShadowBuilder 或重写其方法。默认情况下，您将获得外观与您作为参数传递的视图相同的拖拽阴影，并且中心点位于用户触摸屏幕的位置。

View.DragShadowBuilder()

> 如果使用此构造函数，则 View.DragShadowBuilder 对象中没有任何可用的视图对象（该字段被设为 null）。 如果使用此构造函数并且没有扩展 View.DragShadowBuilder 或重写其方法，您将获得不可见的拖拽阴影。 系统*不会*给出错误。

View.DragShadowBuilder 类有两个方法：

onProvideShadowMetrics()

> 系统会在您调用 startDrag() 后立即调用此方法。使用此方法将拖拽阴影的尺寸和触摸点发送给系统。 此方法具有两个参数：
>
> *dimensions*
>
> > 一个 Point 对象。拖拽阴影的宽度存储在 x 中，高度存储在 y 中。
>
> *touch_point*
>
> > 一个 Point 对象。触摸点是在拖拽操作期间，拖拽阴影内应该处于用户手指下面的位置。 其 X 位置存储在 x 中，Y 位置存储在 y 中。

onDrawShadow()

> 在调用 onProvideShadowMetrics() 之后，系统会立即调用 onDrawShadow() 以获得拖拽阴影本身。 该方法只有一个参数，即系统从您在 onProvideShadowMetrics() 中提供的参数构建的 Canvas 对象 使用此方法在提供的 Canvas 对象中绘制拖拽阴影。

为提高性能，应保持较小的拖拽阴影大小。对于单一项，您可能希望使用图标。 对于多项选择，您可能希望使用堆栈中的图标，而不是在屏幕上展开完整的图像。

设计拖放操作

本部分展示有关如何开始拖拽、如何在拖拽期间响应事件、如何响应放下事件以及如何结束拖放操作的分步说明。

开始拖拽

用户使用拖拽手势（通常是长按视图对象）开始拖拽。 作为响应，您应该执行以下操作：

1. 如果有必要，为正在移动的数据创建 ClipData 和 ClipData.Item。作为 ClipData 对象的一部分，提供存储在 ClipData 内的 ClipDescription 对象中的元数据。 对于不提供数据移动的拖放操作，您可能需要使用 null 而不是实际对象。

例如，以下代码片段显示了如何通过创建包含 ImageView 标记或标签的 ClipData 对象来响应对 ImageView 的长按操作。 此片段之后的下一个片段显示了如何重写 `View.DragShadowBuilder` 中的方法：

```
// Create a string for the ImageView label
private static final String IMAGEVIEW_TAG = "icon bitmap"

// Creates a new ImageView
ImageView imageView = new ImageView(this);

// Sets the bitmap for the ImageView from an icon bit map (defined elsewhere)
imageView.setImageBitmap(mIconBitmap);

// Sets the tag
imageView.setTag(IMAGEVIEW_TAG);

    ...

// Sets a long click listener for the ImageView using an anonymous listener object that
// implements the OnLongClickListener interface
imageView.setOnLongClickListener(new View.OnLongClickListener() {

    // Defines the one method for the interface, which is called when the View is long-clicked
    public boolean onLongClick(View v) {

    // Create a new ClipData.
    // This is done in two steps to provide clarity. The convenience method
    // ClipData.newPlainText() can create a plain text ClipData in one step.

    // Create a new ClipData.Item from the ImageView object's tag
    ClipData.Item item = new ClipData.Item(v.getTag());

    // Create a new ClipData using the tag as a label, the plain text MIME type, and
    // the already-created item. This will create a new ClipDescription object within the
    // ClipData, and set its MIME type entry to "text/plain"
    ClipData dragData = new ClipData(v.getTag(),ClipData.MIMETYPE_TEXT_PLAIN,item);

    // Instantiates the drag shadow builder.
    View.DragShadowBuilder myShadow = new MyDragShadowBuilder(imageView);

    // Starts the drag

            v.startDrag(dragData,  // the data to be dragged
                        myShadow,  // the drag shadow builder
                        null,      // no need to use local data
                        0          // flags (not currently used, set to 0)
            );

    }
}
```

2. 以下代码片段定义了 `myDragShadowBuilder` 它会创建灰色小方框形式的拖拽阴影用于拖拽 TextView：

```
    private static class MyDragShadowBuilder extends View.DragShadowBuilder {

    // The drag shadow image, defined as a drawable thing
    private static Drawable shadow;

        // Defines the constructor for myDragShadowBuilder
        public MyDragShadowBuilder(View v) {

            // Stores the View parameter passed to myDragShadowBuilder.
            super(v);

            // Creates a draggable image that will fill the Canvas provided by the system.
            shadow = new ColorDrawable(Color.LTGRAY);
        }

        // Defines a callback that sends the drag shadow dimensions and touch point back to the
        // system.
        @Override
        public void onProvideShadowMetrics (Point size, Point touch) {
            // Defines local variables
            private int width, height;

            // Sets the width of the shadow to half the width of the original View
            width = getView().getWidth() / 2;

            // Sets the height of the shadow to half the height of the original View
            height = getView().getHeight() / 2;

            // The drag shadow is a ColorDrawable. This sets its dimensions to be the same as the
            // Canvas that the system will provide. As a result, the drag shadow will fill the
            // Canvas.
            shadow.setBounds(0, 0, width, height);

            // Sets the size parameter's width and height values. These get back to the system
            // through the size parameter.
            size.set(width, height);

            // Sets the touch point's position to be in the middle of the drag shadow
            touch.set(width / 2, height / 2);
        }

        // Defines a callback that draws the drag shadow in a Canvas that the system constructs
        // from the dimensions passed in onProvideShadowMetrics().
        @Override
        public void onDrawShadow(Canvas canvas) {

            // Draws the ColorDrawable in the Canvas passed in from the system.
            shadow.draw(canvas);
        }
    }
```

> **注：**请记住，您无需扩展 `View.DragShadowBuilder`。构造函数 `View.DragShadowBuilder(View)` 以拖拽阴影的中心为触摸点，创建与传递给它的视图参数大小相同的默认拖拽阴影。

响应拖拽开始

在拖拽操作期间，系统会将拖拽事件分发到当前布局中的视图对象的拖拽事件侦听器。 侦听器应通过调用 `getAction()` 做出反应，以获取操作类型。 拖拽开始时，此方法将返回 `ACTION_DRAG_STARTED`。

在响应具有操作类型 `ACTION_DRAG_STARTED` 的事件时，侦听器应执行以下操作：

1. 调用 `getClipDescription()` 以获取 `ClipDescription`。使用 `ClipDescription` 中 MIME 类型的方法查看侦听器能否接受正在拖拽的数据。

   如果拖放操作不表示数据移动，这可能不是必要的。

2. 如果侦听器可接受放下操作，则应返回 `true`。这将告知系统继续向侦听器发送拖拽事件。 如果它不能接受放下操作，则应返回 `false`，系统将停止发送拖拽事件，直至其发出 `ACTION_DRAG_ENDED`。

请注意，对于 ACTION_DRAG_STARTED 事件，以下这些 DragEvent 方法全部无效： getClipData()、getX()、getY() 和 getResult()。

在拖拽过程中处理事件

在拖拽期间，返回 true 以响应 ACTION_DRAG_STARTED 拖拽事件的侦听器会继续接收拖拽事件。 侦听器在拖拽期间接收的拖拽事件类型取决于拖拽阴影的位置和侦听器视图的可见性。

在拖拽期间，侦听器主要使用拖拽事件来确定其是否应更改其视图的外观。

在拖拽期间，getAction() 将返回以下三个值中的一个：

- ACTION_DRAG_ENTERED： 侦听器在触摸点（用户手指下面的屏幕点）进入侦听器视图的边界框时收到此事件。
- ACTION_DRAG_LOCATION：在侦听器收到 ACTION_DRAG_ENTERED 事件之后以及收到 ACTION_DRAG_EXITED 事件之前，它会在触摸点每次移动时收到新的 ACTION_DRAG_LOCATION 事件。 getX() 和 getY() 方法会返回触摸点的 X 和 Y 坐标。
- ACTION_DRAG_EXITED：此事件在拖拽阴影不再在侦听器视图的边界框内之后，被发送到之前收到 ACTION_DRAG_ENTERED 的侦听器。

该侦听器不需要对以上任何操作类型做出反应。如果侦听器向系统返回值，该值将被忽略。 以下是响应上述各个操作类型时的一些准则：

- 在响应 ACTION_DRAG_ENTERED 或 ACTION_DRAG_LOCATION 时，侦听器可以更改视图的外观，以指示它将要接收放下操作。
- 具有操作类型 ACTION_DRAG_LOCATION 的事件包含对应于触摸点位置的 getX() 和 getY() 的有效数据。 侦听器可能希望使用此信息来更改位于触摸点的视图部分外观。 侦听器也可以使用此信息来确定用户计划将拖拽阴影拖到的确切位置。
- 在响应 ACTION_DRAG_EXITED 时，侦听器应重置其在响应 ACTION_DRAG_ENTERED 或 ACTION_DRAG_LOCATION 时所应用的任何外观更改。 这向用户指明，该视图不再是迫在眉睫的放下目标。

响应放下

当用户将拖拽阴影释放到应用中的某个视图上并且该视图之前已报告它能接受所拖拽的内容时，系统会向该视图分发具有操作类型 ACTION_DROP 的拖拽事件。 侦听器应执行以下操作：

1. 调用 getClipData() 以获取最初在 startDrag() 调用中提供的 ClipData 对象并存储该对象。 如果拖放操作不表示数据移动，这可能不是必要的。
2. 返回布尔值 true 指示已成功处理放下操作，或者，如果处理失败，则返回布尔值 false。 返回的值将成为 getResult() 针对 ACTION_DRAG_ENDED 事件返回的值。

   请注意，如果系统没有发出 ACTION_DROP 事件，则 ACTION_DRAG_ENDED 事件的 getResult() 值为 false。

对于 ACTION_DROP 事件，getX() 和 getY() 将使用收到放下操作的视图的坐标系，返回拖拽点在放下时刻的 X 和 Y 位置。

系统允许用户将拖拽阴影释放到侦听器未接收拖拽事件的视图上。 它也允许用户在空的应用 UI 区域或在应用以外的区域释放拖拽阴影。 上述所有情况下，系统都不会发送具有操作类型 ACTION_DROP 的事件，不过它会发出 ACTION_DRAG_ENDED 事件。

响应拖拽结束

系统会在用户释放拖放阴影后，立即向应用中的所有拖拽事件侦听器发送具有操作类型 ACTION_DRAG_ENDED 的拖拽事件。 此事件指示拖拽操作结束。

每个侦听器应执行以下操作：

1. 如果侦听器在操作期间更改了其视图对象的外观，则应该将视图重置为其默认外观。 这是通知用户操作结束的视觉指示。
2. 侦听器可以选择调用 getResult() 以了解关于该操作的更多信息。 如果侦听器在响应 ACTION_DROP 操作类型的事件时返回了 true，则 getResult() 将返回布尔值 true。 在其他所有情况下，getResult() 均返回布尔值 false，包括系统未发出 ACTION_DROP 事件的任何情况。
3. 侦听器应该向系统返回布尔值 true。

响应拖拽事件：示例

所有拖拽事件最初都由拖拽事件方法或侦听器接收。以下代码片段是在侦听器中对拖拽事件做出反应的简单示例：

```
// Creates a new drag event listener
mDragListen = new myDragEventListener();

View imageView = new ImageView(this);

// Sets the drag event listener for the View
imageView.setOnDragListener(mDragListen);

...

protected class myDragEventListener implements View.OnDragListener {

    // This is the method that the system calls when it dispatches a drag event to the
    // listener.
    public boolean onDrag(View v, DragEvent event) {

        // Defines a variable to store the action type for the incoming event
        final int action = event.getAction();

        // Handles each of the expected events
        switch(action) {

            case DragEvent.ACTION_DRAG_STARTED:

                // Determines if this View can accept the dragged data
                if (event.getClipDescription().hasMimeType(ClipDescription.MIMETYPE_TEXT_PLAIN)) {

                    // As an example of what your application might do,
                    // applies a blue color tint to the View to indicate that it can accept
                    // data.
                    v.setColorFilter(Color.BLUE);

                    // Invalidate the view to force a redraw in the new tint
                    v.invalidate();

                    // returns true to indicate that the View can accept the dragged data.
                    return true;

                }

                // Returns false. During the current drag and drop operation, this View will
                // not receive events again until ACTION_DRAG_ENDED is sent.
                return false;

            case DragEvent.ACTION_DRAG_ENTERED:

                // Applies a green tint to the View. Return true; the return value is ignored.

                v.setColorFilter(Color.GREEN);

                // Invalidate the view to force a redraw in the new tint
                v.invalidate();

                return true;

            case DragEvent.ACTION_DRAG_LOCATION:

                // Ignore the event
                return true;

            case DragEvent.ACTION_DRAG_EXITED:

                // Re-sets the color tint to blue. Returns true; the return value is ignored.
                v.setColorFilter(Color.BLUE);

                // Invalidate the view to force a redraw in the new tint
                v.invalidate();
```

```
                return true;

            case DragEvent.ACTION_DROP:

                // Gets the item containing the dragged data
                ClipData.Item item = event.getClipData().getItemAt(0);

                // Gets the text data from the item.
                dragData = item.getText();

                // Displays a message containing the dragged data.
                Toast.makeText(this, "Dragged data is " + dragData, Toast.LENGTH_LONG);

                // Turns off any color tints
                v.clearColorFilter();

                // Invalidates the view to force a redraw
                v.invalidate();

                // Returns true. DragEvent.getResult() will return true.
                return true;

            case DragEvent.ACTION_DRAG_ENDED:

                // Turns off any color tinting
                v.clearColorFilter();

                // Invalidates the view to force a redraw
                v.invalidate();

                // Does a getResult(), and displays what happened.
                if (event.getResult()) {
                    Toast.makeText(this, "The drop was handled.", Toast.LENGTH_LONG);

                } else {
                    Toast.makeText(this, "The drop didn't work.", Toast.LENGTH_LONG);

                }

                // returns true; the value is ignored.
                return true;

            // An unknown action type was received.
            default:
                Log.e("DragDrop Example","Unknown action type received by OnDragListener.");
                break;
        }

        return false;
    }
};
```