Drag and Drop

With the Android drag/drop framework, you can allow your users to move data from one View to another View in the current layout using a graphical drag and drop gesture. The framework includes a drag event class, drag listeners, and helper methods and classes.

Although the framework is primarily designed for data movement, you can use it for other UI actions. For example, you could create an app that mixes colors when the user drags a color icon over another icon. The rest of this topic, however, describes the framework in terms of data movement.

Overview

A drag and drop operation starts when the user makes some gesture that you recognize as a signal to start dragging data. In response, your application tells the system that the drag is starting. The system calls back to your application to get a representation of the data being dragged. As the user's finger moves this representation (a "drag shadow") over the current layout, the system sends drag events to the drag event listener objects and drag event callback methods associated with the View (/reference/android/view/View.html) objects in the layout. Once the user releases the drag shadow, the system ends the drag operation.

You create a drag event listener object ("listeners") from a class that implements View.OnDragListener (/reference/android/view/View.OnDragListener.html). You set the drag event listener object for a View with the View object's setOnDragListener()

QUICKVIEW

- Allow users to move data within your Activity layout using graphical gestures.
- Supports operations besides data movement.
- Only works within a single application.
- Requires API 11.

IN THIS DOCUMENT

- Overview
  - The drag/drop process
  - The drag event listener and callback method
  - Drag events
  - The drag shadow
- Designing a Drag and Drop Operation
  - Starting a drag
  - Responding to a drag start
  - Handling events during the drag
  - Responding to a drop
  - Responding to a drag end
  - Responding to drag events: an example

KEY CLASSES

View
OnLongClickListener
OnDragListener
DragEvent
DragShadowBuilder
ClipData
ClipDescription

RELATED SAMPLES

Honeycomb Gallery. DragAndDropDemo.java and DraggableDot.java in Api Demos.

SEE ALSO

Content Providers
Input Events

(/reference/android/view/View.html#setOnDragListener(android.view.View.OnDragListener)) method. Each View object also has a onDragEvent() (/reference/android/view/View.html#onDragEvent(android.view.DragEvent)) callback method. Both of these are described in more detail in the section The drag event listener and callback method (#AboutDragListeners).

> Note: For the sake of simplicity, the following sections refer to the routine that receives drag events as the "drag event listener", even though it may actually be a callback method.

When you start a drag, you include both the data you are moving and metadata describing this data as part of the call to the system. During the drag, the system sends drag events to the drag event listeners or callback methods of each View in the layout. The listeners or callback methods can use the metadata to decide if they want to accept the data when it is dropped. If the user drops the data over a View object, and that View object's listener or callback method has previously told the system that it wants to accept the drop, then the system sends the data to the listener or callback method in a drag event.

Your application tells the system to start a drag by calling the startDrag() (/reference/android/view/View.html#startDrag(android.content.ClipData, android.view.View.DragShadowBuilder, java.lang.Object, int)) method. This tells the system to start sending drag events. The method also sends the data that you are dragging.

You can call startDrag() (/reference/android/view/View.html#startDrag(android.content.ClipData, android.view.View.DragShadowBuilder, java.lang.Object, int)) for any attached View in the current layout. The system only uses the View object to get access to global settings in your layout.

Once your application calls startDrag() (/reference/android/view/View.html#startDrag(android.content.ClipData, android.view.View.DragShadowBuilder, java.lang.Object, int)), the rest of the process uses events that the system sends to the View objects in your current layout.

The drag/drop process

There are basically four steps or states in the drag and drop process:

Started

In response to the user's gesture to begin a drag, your application calls startDrag() to tell the system to start a drag. The arguments startDrag() provide the data to be dragged, metadata for this data, and a callback for drawing the drag shadow.

The system first responds by calling back to your application to get a drag shadow. It then displays the drag shadow on the device.

Next, the system sends a drag event with action type ACTION_DRAG_STARTED (/reference/android/view/DragEvent.html#ACTION_DRAG_STARTED) to the drag event listeners for all the View objects in the current layout. To continue to receive drag events, including a possible drop event, a drag event listener must return `true`. This registers the listener with the system. Only registered listeners continue to receive drag events. At this point, listeners can also change the appearance of their View object to show that the listener can accept a drop event.

If the drag event listener returns `false`, then it will not receive drag events for the current operation until the system sends a drag event with action type ACTION_DRAG_ENDED (/reference/android/view/DragEvent.html#ACTION_DRAG_ENDED). By sending `false`, the listener tells the system that it is not interested in the drag operation and does not want to accept the dragged data.

Continuing

The user continues the drag. As the drag shadow intersects the bounding box of a View object, the system sends one or more drag events to the View object's drag event listener (if it is registered to receive events). The listener may choose to alter its View object's appearance in response to the event. For example, if the event indicates that the drag shadow has entered the bounding box of the View (action type ACTION_DRAG_ENTERED), the listener can react by highlighting its View.

Dropped

The user releases the drag shadow within the bounding box of a View that can accept the data. The system sends the View object's listener a drag event with action type ACTION_DROP. The drag event contains the data that was passed to the system in the call to startDrag() that started the operation. The listener is expected to return boolean true to the system if code for accepting the drop succeeds.

Note that this step only occurs if the user drops the drag shadow within the bounding box of a View whose listener is registered to receive drag events. If the user releases the drag shadow in any other situation, no ACTION_DROP (/reference/android/view/DragEvent.html#ACTION_DROP) drag event is sent.

Ended

After the user releases the drag shadow, and after the system sends out (if necessary) a drag event with action type ACTION_DROP, the system sends out a drag event with action type ACTION_DRAG_ENDED to indicate that the drag operation is over. This is done regardless of where the user released the drag shadow. The event is sent to every listener that is registered to receive drag events, even if the listener received the ACTION_DROP event.

Each of these four steps is described in more detail in the section Designing a Drag and Drop Operation (#DesignDragOperation).

The drag event listener and callback method

A View receives drag events with either a drag event listener that implements View.OnDragListener (/reference/android/view/View.OnDragListener.html) or with its onDragEvent(DragEvent) (/reference/android/view/View.html#onDragEvent(android.view.DragEvent)) callback method. When the system calls the method or listener, it passes to them a DragEvent (/reference/android/view/DragEvent.html) object.

You will probably want to use the listener in most cases. When you design UIs, you usually don't subclass View classes, but using the callback method forces you to do this in order to override the method. In comparison, you can implement one listener class and then use it with several different View objects. You can also implement it as an anonymous inline class. To set the listener for a View object, call setOnDragListener() (/reference/android/view/View.html#setOnDragListener(android.view.View.OnDragListener)).

You can have both a listener and a callback method for View object. If this occurs, the system first calls the listener. The system doesn't call the callback method unless the listener returns false.

The combination of the onDragEvent(DragEvent) (/reference/android/view/View.html#onDragEvent(android.view.DragEvent)) method and View.OnDragListener (/reference/android/view/View.OnDragListener.html) is analogous to the combination of the onTouchEvent() (/reference/android/view/View.html#onTouchEvent(android.view.MotionEvent)) and View.OnTouchListener (/reference/android/view/View.OnTouchListener.html) used with touch events.

Drag events

The system sends out a drag event in the form of a DragEvent (/reference/android/view/DragEvent.html) object. The object contains an action type that tells the listener what is happening in the drag/drop process. The object contains other data, depending on the action type.

To get the action type, a listener calls getAction() (/reference/android/view/DragEvent.html#getAction()). There are six possible values, defined by constants in the DragEvent (/reference/android/view/DragEvent.html) class. These are listed in table 1 (#table1).

The DragEvent (/reference/android/view/DragEvent.html) object also contains the data that your application provided to the system in the call to startDrag() (/reference/android/view/View.html#startDrag(android.content.ClipData, android.view.View.DragShadowBuilder, java.lang.Object, int)). Some of the data is valid only for certain action types. The data that is valid for each action type is summarized in table 2 (#table2). It is also described in detail with the event for which it is valid in the section Designing a Drag and Drop Operation (#DesignDragOperation).

Table 1. DragEvent action types

getAction() value	Meaning
ACTION_DRAG_STARTED	A View object's drag event listener receives this event action type just after the

	application calls startDrag() and gets a drag shadow.
ACTION_DRAG_ENTERED	A View object's drag event listener receives this event action type when the drag shadow has just entered the bounding box of the View. This is the first event action type the listener receives when the drag shadow enters the bounding box. If the listener wants to continue receiving drag events for this operation, it must return boolean true to the system.
ACTION_DRAG_LOCATION	A View object's drag event listener receives this event action type after it receives a ACTION_DRAG_ENTERED event while the drag shadow is still within the bounding box of the View.
ACTION_DRAG_EXITED	A View object's drag event listener receives this event action type after it receives a ACTION_DRAG_ENTERED and at least one ACTION_DRAG_LOCATION event, and after the user has moved the drag shadow outside the bounding box of the View.
ACTION_DROP	A View object's drag event listener receives this event action type when the user releases the drag shadow over the View object. This action type is only sent to a View object's listener if the listener returned boolean true in response to the ACTION_DRAG_STARTED drag event. This action type is not sent if the user releases the drag shadow on a View whose listener is not registered, or if the user releases the drag shadow on anything that is not part of the current layout. The listener is expected to return boolean true if it successfully processes the drop. Otherwise, it should return false.
ACTION_DRAG_ENDED	A View object's drag event listener receives this event action type when the system is ending the drag operation. This action type is not necessarily preceded by an ACTION_DROP event. If the system sent a ACTION_DROP, receiving the ACTION_DRAG_ENDED action type does not imply that the drop operation succeeded. The listener must call getResult() to get the value that was returned in response to ACTION_DROP. If an ACTION_DROP event was not sent, then getResult() returns false.

Table 2. Valid DragEvent data by action type

getAction() value	getClipDescription() value	getLocalState() value	getX() value	getY() value	getClipData() value	getResult() value
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

The getAction() (/reference/android/view/DragEvent.html#getAction()), describeContents() (/reference/android/view/DragEvent.html#describeContents()), writeToParcel() (/reference/android/view/DragEvent.html#writeToParcel(android.os.Parcel, int)), and toString() (/reference/android/view/DragEvent.html#toString()) methods always return valid data.

If a method does not contain valid data for a particular action type, it returns either null or 0, depending on its result type.

The drag shadow

During a drag and drop operation, the system displays a image that the user drags. For data movement, this image represents the data being dragged. For other operations, the image represents some aspect of the drag operation.

The image is called a drag shadow. You create it with methods you declare for a View.DragShadowBuilder (/reference/android/view/View.DragShadowBuilder.html) object, and then pass it to the system when you start a drag using startDrag() (/reference/android/view/View.html#startDrag(android.content.ClipData, android.view.View.DragShadowBuilder, java.lang.Object, int)). As part of its response to startDrag() (/reference/android/view/View.html#startDrag(android.content.ClipData, android.view.View.DragShadowBuilder, java.lang.Object, int)), the system invokes the callback methods you've defined in View.DragShadowBuilder

(/reference/android/view/View.DragShadowBuilder.html) to obtain a drag shadow.

The View.DragShadowBuilder (/reference/android/view/View.DragShadowBuilder.html) class has two constructors:

View.DragShadowBuilder(View)
: This constructor accepts any of your application's View objects. The constructor stores the View object in the View.DragShadowBuilder object, so during the callback you can access it as you construct your drag shadow. It doesn't have to be associated with the View (if any) that the user selected to start the drag operation.

  If you use this constructor, you don't have to extend View.DragShadowBuilder (/reference/android/view/View.DragShadowBuilder.html) or override its methods. By default, you will get a drag shadow that has the same appearance as the View you pass as an argument, centered under the location where the user is touching the screen.

View.DragShadowBuilder()
: If you use this constructor, no View object is available in the View.DragShadowBuilder object (the field is set to null). If you use this constructor, and you don't extend View.DragShadowBuilder or override its methods, you will get an invisible drag shadow. The system does *not* give an error.

The View.DragShadowBuilder (/reference/android/view/View.DragShadowBuilder.html) class has two methods:

onProvideShadowMetrics()
: The system calls this method immediately after you call startDrag(). Use it to send to the system the dimensions and touch point of the drag shadow. The method has two arguments:

  dimensions
  : A Point object. The drag shadow width goes in x and its height goes in y.

  touch_point
  : A Point object. The touch point is the location within the drag shadow that should be under the user's finger during the drag. Its X position goes in x and its Y position goes in y

onDrawShadow()
: Immediately after the call to onProvideShadowMetrics() the system calls onDrawShadow() to get the drag shadow itself. The method has a single argument, a Canvas object that the system constructs from the parameters you provide in onProvideShadowMetrics() Use it to draw the drag shadow in the provided Canvas object.

To improve performance, you should keep the size of the drag shadow small. For a single item, you may want to use a icon. For a multiple selection, you may want to use icons in a stack rather than full images spread out over the screen.

Designing a Drag and Drop Operation

This section shows step-by-step how to start a drag, how to respond to events during the drag, how respond to a drop event, and how to end the drag and drop operation.

Starting a drag

The user starts a drag with a drag gesture, usually a long press, on a View object. In response, you should do the following:

1. As necessary, create a ClipData and ClipData.Item for the data being moved. As part of the ClipData object, supply metadata that is stored in a ClipDescription object within the ClipData. For a drag and drop operation that does not represent data movement, you may want to use null instead of an actual object.

   For example, this code snippet shows how to respond to a long press on a ImageView by creating a ClipData object that contains the tag or label of an ImageView. Following this snippet, the next snippet shows how to override the methods in View.DragShadowBuilder (/reference/android/view/View.DragShadowBuilder.html):

```
// 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-click
    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. The following code snippet defines `myDragShadowBuilder` It creates a drag shadow for dragging a TextView as a small gray rectangle:

```
    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
        // 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 a
            // Canvas that the system will provide. As a result, the drag shadow will fill t
            // Canvas.
            shadow.setBounds(0, 0, width, height);

            // Sets the size parameter's width and height values. These get back to the syst
            // 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 constru
        // 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);
        }
    }
```

> Note: Remember that you don't have to extend View.DragShadowBuilder (/reference/android/view/View.DragShadowBuilder.html). The constructor View.DragShadowBuilder(View) (/reference/android/view/View.DragShadowBuilder.html#View.DragShadowBuilder(android.view.View)) creates a default drag shadow that's the same size as the View argument passed to it, with the touch point centered in the drag shadow.

Responding to a drag start

During the drag operation, the system dispatches drag events to the drag event listeners of the View objects in the current layout. The listeners should react by calling getAction() (/reference/android/view/DragEvent.html#getAction()) to get the action type. At the start of a drag, this methods returns ACTION_DRAG_STARTED (/reference/android/view/DragEvent.html#ACTION_DRAG_STARTED).

In response to an event with the action type ACTION_DRAG_STARTED (/reference/android/view/DragEvent.html#ACTION_DRAG_STARTED), a listener should do the following:

1. Call getClipDescription() to get the ClipDescription. Use the MIME type methods in ClipDescription to see if the listener can accept the data being dragged.

If the drag and drop operation does not represent data movement, this may not be necessary.

2. If the listener can accept a drop, it should return `true`. This tells the system to continue to send drag events to the listener. If it can't accept a drop, it should return `false`, and the system will stop sending drag events until it sends out ACTION_DRAG_ENDED.

Note that for an ACTION_DRAG_STARTED (/reference/android/view/DragEvent.html#ACTION_DRAG_STARTED) event, these the following DragEvent (/reference/android/view/DragEvent.html) methods are not valid: getClipData() (/reference/android/view/DragEvent.html#getClipData()), getX() (/reference/android/view/DragEvent.html#getX()), getY() (/reference/android/view/DragEvent.html#getY()), and getResult() (/reference/android/view/DragEvent.html#getResult()).

During the drag, listeners that returned `true` in response to the ACTION_DRAG_STARTED (/reference/android/view/DragEvent.html#ACTION_DRAG_STARTED) d… …vent continue to receive drag events. The types of drag events a listener receives during the drag depend o… …ocation of the drag shadow and the visibility of the listener's View.

App Components
App Resources
App Manifest
User Interface
Overview
Layouts
Input Controls
Input Events
Menus
Action Bar
Settings
Dialogs
Notifications
Toasts
Search
Drag and Drop
Accessibility
Styles and Themes

During the drag, listeners primarily use drag events to deci… …ney should change the appearance of their View.

During the drag, getAction() … …tml#getAction()) returns one of three values:

- ACTION_DRAG_ENTERED: The list … …point (the point on the screen underneath the user's finger) has entered the … …'.
- ACTION_DRAG_LOCATION: Once t… …_ENTERED event, and before it receives an AACTION_DRAG_EXITED event, it… …TION event every time the touch point moves. The getX() and getY() metho… …f the touch point.
- ACTION_DRAG_EXITED: This eve… …ly received ACTION_DRAG_ENTERED, after the drag shadow is no longer with… …s View.

The listener does not need to … …f the listener returns a value to the system, it is ignored. Here are some guide… …e action types:

- In response to ACTION_DRAG_E… …e listener can change the appearance of the View to indicate that it is abou…
- An event with the action type… …id data for getX() and getY(), corresponding to the location of the touch pc… …is information to alter the appearance of that part of the View that is at the … …se this information to determine the exact position where the user is goi…
- In response to ACTION_DRAG_E… …appearance changes it applied in response to ACTION_DRAG_ENTERED or ACTIC… …the user that the View is no longer an imminent drop target.

Responding to a drop

When the user releases the drag shadow on a View in the application, and that View previously reported that it could accept the content being dragged, the system dispatches a drag event to that View with the action type ACTION_DROP (/reference/android/view/DragEvent.html#ACTION_DROP). The listener should do the following:

1. Call getClipData() to get the ClipData object that was originally supplied in the call to startDrag() and store it. If the drag and drop operation does not represent data movement, this may not be necessary.
2. Return boolean `true` to indicate that the drop was processed successfully, or boolean `false` if it was not. The returned value becomes the value returned by getResult() for an ACTION_DRAG_ENDED event.

   Note that if the system does not send out an ACTION_DROP (/reference/android/view/DragEvent.html#ACTION_DROP) event, the value of getResult() (/reference/android/view/DragEvent.html#getResult()) for an ACTION_DRAG_ENDED (/reference/android/view/DragEvent.html#ACTION_DRAG_ENDED) event is `false`.

For an ACTION_DROP (/reference/android/view/DragEvent.html#ACTION_DROP) event, getX() (/reference/android/view/DragEvent.html#getX()) and getY() (/reference/android/view/DragEvent.html#getY()) return the X and Y position of the drag point at the moment of the drop, using the coordinate system of the View that

received the drop.

The system does allow the user to release the drag shadow on a View whose listener is not receiving drag events. It will also allow the user to release the drag shadow on empty regions of the application's UI, or on areas outside of your application. In all of these cases, the system does not send an event with action type ACTION_DROP (/reference/android/view/DragEvent.html#ACTION_DROP), although it does send out an ACTION_DRAG_ENDED (/reference/android/view/DragEvent.html#ACTION_DRAG_ENDED) event.

Responding to a drag end

Immediately after the user releases the drag shadow, the system sends a drag event to all of the drag event listeners in your application, with an action type of ACTION_DRAG_ENDED (/reference/android/view/DragEvent.html#ACTION_DRAG_ENDED). This indicates that the drag operation is over.

Each listener should do the following:

1. If listener changed its View object's appearance during the operation, it should reset the View to its default appearance. This is a visual indication to the user that the operation is over.
2. The listener can optionally call getResult() to find out more about the operation. If a listener returned `true` in response to an event of action type ACTION_DROP, then getResult() will return boolean `true`. In all other cases, getResult() returns boolean `false`, including any case in which the system did not send out a ACTION_DROP event.
3. The listener should return boolean `true` to the system.

Responding to drag events: an example

All drag events are initially received by your drag event method or listener. The following code snippet is a simple example of reacting to drag events in a listener:

```
// 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;
    }
};
```