JavaFX

Drag-and-Drop Feature in JavaFX Applications

Release 2.1

E24176-05

June 2013

ORACLE

JavaFX/Drag-and-Drop Feature in JavaFX Applications, Release 2.1

E24176-05

Primary Author: Irina Fedortsova

Contributor: Pavel Safrata

Contents

1 Drag-and-Drop Feature in JavaFX Applications

1

Drag-and-Drop Feature in JavaFX Applications

In this article, you learn how to implement a drag-and-drop feature in JavaFX 2 applications, which objects participate in a drag-and-drop gesture, what types of data can be transferred, and which events occur during a drag-and-drop gesture.

This article also includes code samples to illustrate the APIs being used and the explained material.

Introduction

A drag-and-drop operation is a data transfer between two objects: a gesture source and a gesture target. The gesture source and gesture target can be the following objects:

- Nodes
- Scenes

The gesture source and gesture target can belong to a single JavaFX application or to two different JavaFX or Java Client applications. Moreover, drag-and-drop can be implemented between a JavaFX application and a third-party (native) application such as Windows Explorer or a desktop.

A drag-and-drop gesture happens as follows: The user click a mouse button on a gesture source, drags the mouse, and releases the mouse button on a gesture target. While dragging the data, the user gets visual feedback, which denotes locations that do not accept the data and, when over a target that accepts the data, gives a hint where to drop the data.

The data is transferred using a dragboard, which has the same interface as a system clipboard but is only used for the drag-and-drop data transfer.

During the drag-and-drop gesture, various types of data can be transferred such as text, images, URLs, files, bytes, and strings.

The `javafx.scene.input.DragEvent` class is the basic class used to implement the drag-and-drop gesture. For more information on particular methods and other classes in the `javafx.scene.input` package, see the API documentation.

Transfer Modes

Transfer modes define the type of transfer that happens between the gesture source and gesture target. Available transfer modes include `COPY`, `MOVE`, and `LINK`.

A gesture source reports supported transfer modes. A gesture target accepts one or more transfer modes. The transfer mode in a given drag-and-drop gesture is chosen by the system from the modes supported by the source and accepted by the target according to the keyboard modifiers pressed by the user.

Implementing a Basic Drag-and-Drop Gesture

You can learn how to implement basic drag-and-drop functionality by using the `HelloDragAndDrop` sample application. To download the source code, click the link in the sidebar. The gesture source and gesture target are two text nodes defined as shown in Example 1–1.

Example 1–1

```
final Text source = new Text(50, 100, "DRAG ME");
final Text target = new Text(300, 100, "DROP HERE");
```

Starting the Drag-and-Drop Gesture on a Source

The drag-and-drop gesture can only be started by calling the `startDragAndDrop` method inside the handler of the `DRAG_DETECTED` event on a gesture source. It is here that transfer modes supported by the gesture source are defined, and the data to be transferred is placed onto the dragboard.

See the implementation of the `onDragDetected` handler in Example 1–2.

Example 1–2

```
source.setOnDragDetected(new EventHandler<MouseEvent>() {
    public void handle(MouseEvent event) {
        /* drag was detected, start a drag-and-drop gesture*/
        /* allow any transfer mode */
        Dragboard db = source.startDragAndDrop(TransferMode.ANY);

        /* Put a string on a dragboard */
        ClipboardContent content = new ClipboardContent();
        content.putString(source.getText());
        db.setContent(content);

        event.consume();
    }
});
```

The `startDragAndDrop` method takes a set of transfer modes supported by the gesture source. You can pass any combination of available transfer modes. By passing `TransferMode.COPY`, you can indicate that the gesture source only supports copying, but not moving or referencing.

Handling a DRAG_OVER Event on a Target

After the drag-and-drop gesture is started, any node or scene that the mouse is dragged over is a potential target to drop the data. You specify which object accepts the data by implementing the `DRAG_OVER` event handler.

Note the importance of the `DRAG_OVER` event handler. For a successful drag-and-drop operation, you must implement the `DRAG_OVER` event handler, which calls the `acceptTransferModes(TransferMode...)` method on the event, passing the transfer modes that the target intends to accept. If none of the passed transfer modes are

supported by the gesture source, the potential target does not fit the given drag-and-drop gesture.

Note that the type of data available on the dragboard must be taken into account when deciding whether to accept the event. To access the data stored on the dragboard, use the `event.getDragboard()` method.

Example 1–3 shows the implementation of the `DRAG_OVER` event handler.

Example 1–3

```
target.setOnDragOver(new EventHandler<DragEvent>() {
    public void handle(DragEvent event) {
        /* data is dragged over the target */
        /* accept it only if it is not dragged from the same node
         * and if it has a string data */
        if (event.getGestureSource() != target &&
                event.getDragboard().hasString()) {
            /* allow for both copying and moving, whatever user chooses */
            event.acceptTransferModes(TransferMode.COPY_OR_MOVE);
        }

        event.consume();
    }
});
```

Providing Visual Feedback by a Gesture Target

During a drag-and-drop gesture, when the mouse pointer hovers over a target that fits the given drag-and-drop gesture, the target typically changes its appearance to provide a hint to the user where the data can be dropped.

When the drag gesture enters the boundaries of a potential gesture target, the target receives a `DRAG_ENTERED` event. When the drag gesture leaves the potential target's boundaries, the target receives a `DRAG_EXITED` event. You can use the `DRAG_ENTERED` and `DRAG_EXITED` event handlers to change the target's appearance in order to provide the visual feedback to the user.

Example 1–4 shows how the visual feedback is implemented by changing the color of the text.

Example 1–4

```
target.setOnDragEntered(new EventHandler<DragEvent>() {
    public void handle(DragEvent event) {
    /* the drag-and-drop gesture entered the target */
    /* show to the user that it is an actual gesture target */
         if (event.getGestureSource() != target &&
                 event.getDragboard().hasString()) {
             target.setFill(Color.GREEN);
         }

         event.consume();
    }
});
```

Note the importance of verifying the contents of the dragboard. The target only changes its appearance if the dragboard contains data in the proper format, which is a string in this case.

Example 1–5 shows the implementation of the DRAG_EXITED event handler, which restores the original appearance of the text.

Example 1–5

```
target.setOnDragExited(new EventHandler<DragEvent>() {
    public void handle(DragEvent event) {
        /* mouse moved away, remove the graphical cues */
        target.setFill(Color.BLACK);

        event.consume();
    }
});
```

Handling a DRAG_DROPPED Event on a Target

When the mouse button is released on the gesture target, which accepted previous DRAG_OVER events with a transfer mode supported by the gesture source, then the DRAG_DROPPED event is sent to the gesture target. In the DRAG_DROPPED event handler, you must complete the drag-and-drop gesture by calling the setDropCompleted(Boolean) method on the event. Otherwise, the gesture is considered unsuccessful.

See the implementation of the DRAG_DROPPED event handler in Example 1–6.

Example 1–6

```
target.setOnDragDropped(new EventHandler<DragEvent>() {
    public void handle(DragEvent event) {
        /* data dropped */
        /* if there is a string data on dragboard, read it and use it */
        Dragboard db = event.getDragboard();
        boolean success = false;
        if (db.hasString()) {
           target.setText(db.getString());
           success = true;
        }
        /* let the source know whether the string was successfully
         * transferred and used */
        event.setDropCompleted(success);

        event.consume();
     }
});
```

Handling a DRAG_DONE Event on a Source

After the drag-and-drop gesture is finished, the DRAG_DONE event is sent to the gesture source to inform the source about how the gesture finished. In the DRAG_DONE event handler, get the transfer mode by calling the getTransferMode method on the event. If the transfer mode is NULL then that means the data transfer did not happen. If the mode is MOVE, then clear the data on the gesture source as shown in Example 1–7.

Example 1–7

```
source.setOnDragDone(new EventHandler<DragEvent>() {
    public void handle(DragEvent event) {
        /* the drag and drop gesture ended */
```

```
        /* if the data was successfully moved, clear it */
        if (event.getTransferMode() == TransferMode.MOVE) {
            source.setText("");
        }
        event.consume();
    }
});
```

Dragging Custom Data

Similarly, you can implement the drag-and-drop gesture on custom data. Define the custom data type as shown in Example 1–8:

Example 1–8

```
/** The custom format */
private static final DataFormat customFormat =
    new DataFormat("helloworld.custom");
```

When putting a custom data onto a dragboard, specify the data type. Note that the data must be serializable.

When reading the data from the dragboard, a proper casting is needed.

View and Download Application Files

View Source Code

http://docs.oracle.com/javafx/2/drag_drop/HelloDragAndDrop.java.html

Download Source Code

http://docs.oracle.com/javafx/2/drag_drop/hellodraganddrop.zip