

Drag and Drop

Vaadin 14.1+

Agenda

- Intro
- Generic drag & drop
- Drag & drop events
- Exercise 1
- Grid drag & drop
- Exercise 2
- Dropping files

What is drag and drop?

Drag and Drop is a few concepts blended together. There are three main cases:

- Generic dragging of components inside a Vaadin application
- Custom drag and drop of items in Grid and related components
- Dragging files from outside the browser into the application

Generic Vaadin D&D

Available since Vaadin **14.1**

Based on the HTML5 Drag and Drop
API.

HTML



How to do D&D with Vaadin components?



Get started

```
Button button = new Button("Hello");
```

```
Div left = new Div(button);  
left.addClassName("dropbox");
```

```
Div right = new Div();  
right.addClassName("dropbox");
```



Enable Dragging

A component can become draggable by wrapping it into a **DragSource** extension

```
//A component is not draggable by default
```

```
Button button = new Button("Hello");
```

```
//A component becomes draggable after being wrapped into a DragSource
```

```
DragSource.create(button);
```

Enable Dragging

An alternative way to enable dragging is to implement the DragSource interface

```
public class MyComponent extends Component implements DragSource<MyComponent> {  
  
}
```


Enable Dropping

A component can accept drops by wrapping it with a **DropTarget** extension

```
//A component is not droppable by default
```

```
Div div = new Div();
```

```
//A component becomes droppable after being wrapped into a DragSource
```

```
DropTarget.create(div);
```

Enable Dropping

An alternative way to enable dropping is to implement the DropTarget interface

```
public class MyComponent extends Component implements DropTarget<MyComponent> {  
}
```

Not yet

Even if you have a drag source and a drop target, Vaadin doesn't move the component to the drop target automatically.

In this case, to move a component between two containers, you need to listen for a drop event and move the component to the drop target when it occurs.

There are many possible ways to use dragging - Vaadin provides drag and drop event listeners as places where you can implement them.



DnD events

DropEvent

DragStartedEvent

DragEndEvent

DropEvent

Usually you need at least an event handler for the drop event. Use **DropTarget** to listen for the event with the **addDropListener()** method.

```
Div right = new Div();  
  
DropTarget<Div> dropTarget = DropTarget.create(right);  
  
dropTarget.addDropListener(event -> {  
  
});
```

DropEvent

From the event, you can get the **source component** that is being dragged and the **target component** that is about to drop to.

```
Div right = new Div();
```

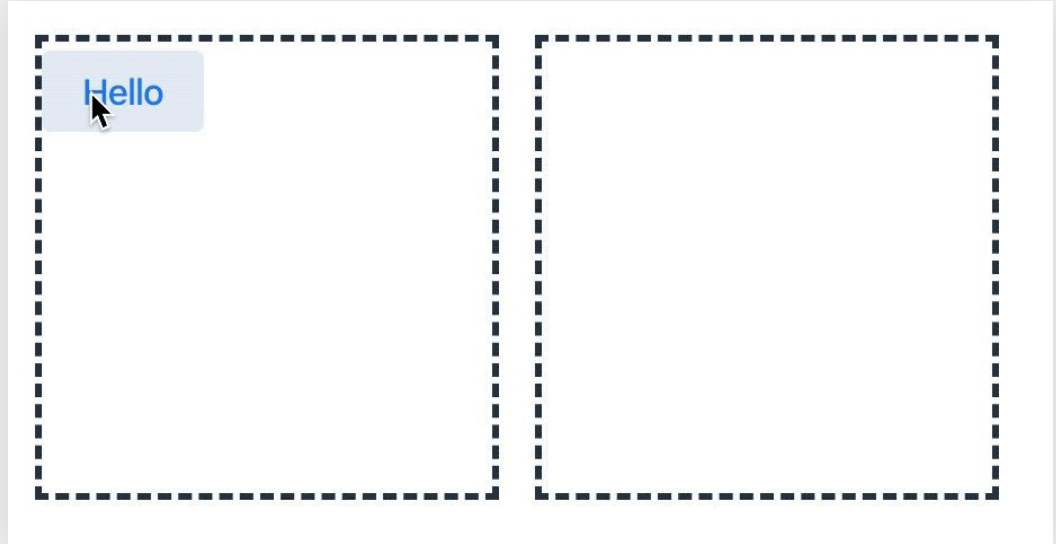
```
DropTarget<Div> dropTarget = DropTarget.create(right);
```

```
dropTarget.addDropListener(event -> {  
    //get the source component that is being dragged  
    Optional<Component> source = event.getDragSourceComponent();  
  
    //get the target component that is about to drop to  
    event.getComponent();  
});
```

Now it works!

```
DropTarget<Div> target = DropTarget.create(right);

target.addDropListener(e ->
    e.getDragSourceComponent().ifPresent(
        component -> target.add(component)
    )
);
```



DragStartEvent

The event is fired when drag starts. Use **DragSource** to listen for the event with the **addDragStartListener()** method.

```
Button button = new Button();  
  
DragSource<Button> dragSource = DragSource.create(button);  
  
dragSource.addDragStartListener(event -> {  
  
});
```


DragStartEvent

From the event, you can get the **source component** that is being dragged.

```
Button button = new Button();

DragSource<Button> dragSource = DragSource.create(button);

dragSource.addDragStartListener(event -> {
    //get the source component that is being dragged
    event.getComponent();
});
```

DragEndEvent

The event is fired when drag ends. Use **DragSource** to listen for the event with the **addDragStartListener()** method.

```
Button button = new Button();  
  
DragSource<Button> dragSource = DragSource.create(button);  
  
dragSource.addDragStartListener(event -> {  
  
});
```

DragEndEvent

From the event, you can get the **source component** that is being dragged.

```
Button button = new Button();

DragSource<Button> dragSource = DragSource.create(button);

dragSource.addDragEndListener(event -> {
    //get the source component that is being dragged
    event.getComponent();
});
```

DnD events

Q: DropEvent vs DragEndEvent, which one comes first?

A: DropEvent comes first, then the DragEndEvent

Drop Effects

You can use a Drop Effect to configure what the cursor looks like when dragging an item on top of a DropTarget. The styles of the cursors are browser-specific. You can also configure a DragSource to only allow specific types (one or more) of effects.

Possible effects are:

- COPY - indicates a copy of the source will be made at the new location
- MOVE - the item is to be moved to the new location
- LINK - the old and new location should be linked somehow
- NONE - the item can't be dropped

Allowed Drop Effects

```
leftDropTarget.setDropEffect(DropEffect.COPY);
```

```
rightDropTarget.setDropEffect(DropEffect.MOVE);
```

```
buttonDragSource.setEffectAllowed(EffectAllowed.MOVE);
```

DropEffect: Copy

Effect allowed: Move



DropEffect: Move

Multiple allowed Drop Effects

```
leftDropTarget.setDropEffect(DropEffect.COPY);
```

```
rightDropTarget.setDropEffect(DropEffect.MOVE);
```

```
buttonDragSource.setEffectAllowed(EffectAllowed.COPY_MOVE);
```

DropEffect: Copy

Effect allowed: Copy or Move



DropEffect: Move

Drop Effects

Note that setting a Drop Effect does not implement the implied action. It's just a hint to the end user of what they may expect to happen and a way to easily disallow drops from certain types of sources.

Exercise





Kanban board

Grid DnD

Grid DnD

- Row reordering inside a Grid
- Drag rows between Grids
- Applies to both Grid and TreeGrid

Row Reordering

First Name 	Last Name 	Phone Number 
Brandon 	Wheeler	904-077-8070
Brayden	Wilder	915-088-178
Brody	Witt	921-646-6501
Brooklyn	Woodward	922-563-797
Bentley	West	878-354-976

Enable Dragging

Grid rows become draggable after calling **setRowsDraggable(true)** on a Grid.

As with generic drag and drop, enabling dragging in itself is not very helpful; you need to implement the wanted action yourself. It could be copying, moving or linking the items.

```
//Enable user to drag grid rows  
grid.setRowsDraggable(true);
```

GridDragStartEvent

Listen for the event with **grid.addDragStartListener()**. From the event, you can get the dragged item (or a list of items, if multiselection is enabled). You'll need it later!

```
private Person draggedItem;

//...

Grid<Person> grid = new Grid<>();
grid.addDragStartListener(event -> {
    // get the dragged item(s)
    draggedItem = event.getDraggedItems().get(0);
    // this is a good place to set the DropMode
    grid.setDropMode(GridDropMode.BETWEEN);
});
```

GridDropEvent

What you want to do when a drop occurs depends e.g. on the DataProvider. To reorder rows within a single Grid with a ListDataProvider (remember that a reference to the dragged item was stored in the DragStartEvent listener):

```
grid.addDropListener( dropEvent -> {
    Person dropOverItem = dropEvent.getDropTargetItem().get();
    if (!dropOverItem.equals(draggedItem)) {
        gridItems.remove(draggedItem);
        int dropIndex = gridItems.indexOf(dropOverItem);
        // see next slide about drop locations
        if (dropEvent.getDropLocation() == GridDropLocation.BELOW) {
            dropIndex++;
        }
        gridItems.add(dropIndex, draggedItem);
        grid.getDataProvider().refreshAll();
    }
});
```



Drop Mode and Drop Location



A Grid's drop mode can be set with `grid.setDropMode()`. The drop mode is used to tell where drops can happen (where the target outline is drawn while dragging) and affects drop event's `DropLocation` value. The drop location tells where the drop has occurred relative to the target. When dropping on an empty Grid or the empty area below the last row, `DropLocation` is `EMPTY`.

Possible `GridDropMode` values:

- `ON_GRID`: Drops occur on the entire Grid. The drop event will not contain row information.
- `BETWEEN`: The drop outline is shown between rows. The `DropLocation` is `ABOVE` a row when the cursor is over the top 50% of a row, otherwise it's `BELOW` the row.
- `ON_TOP`: The drop outline surrounds a row. The `DropLocation` is `ON_TOP`.
- `ON_TOP_OR_BETWEEN`: A combination of the two options above. `DropLocation` can be either `ABOVE`, `BELOW` or `ON_TOP` of a row.

Dragging Rows Between Grids

<input type="checkbox"/>	First Name 	Last Name 
<input type="checkbox"/>	Bentley	West
<input type="checkbox"/>	Brandon	Wheeler
<input type="checkbox"/>	Brayden	Wilder
<input type="checkbox"/>	Brody	Witt
<input type="checkbox"/>	Brooklyn	Woodward

<input type="checkbox"/>	First Name 	Last Name 
<input type="checkbox"/>	Caleb	Yates

Dragging Rows Between Grids

When moving rows between Grids, the same principle applies as when reordering rows within a single Grid.

- Keep track of what is being dragged in a DragStartEvent listener
- When a drop occurs - DropListener:
 - Remove the items from the original DataProvider's backing data source
 - Calculate the position where the new items should be added based on the drop target item
 - Insert the items into the target data source
 - Remember to refresh both DataProviders
- See a full example in Grid's component documentation:
<https://vaadin.com/components/vaadin-grid/java-examples/drag-and-drop>

Copying a row from one Grid to another works similarly, except instead of removing the items from the source, you should create new instances to the target data source.

Exercise 2

Drop files?

Use the [Upload](#) component

Upload File...



Drop file here

```
MemoryBuffer buffer = new MemoryBuffer();
Upload upload = new Upload(buffer);

upload.addSucceededListener(event -> {
    Component component = createComponent(event.getMIMEType(),
        event.getFileName(), buffer.getInputStream());
    showOutput(event.getFileName(), component, output);
});
```

Drop files?

Upload supports single or multiple-file uploads via drag and drop (or file browser). Uploaded files can be stored in memory or directly to the file system. To select this behavior, choose a suitable built in **Receiver**:

- **FileBuffer**
- **MemoryBuffer**
- **MultiFileBuffer**
- **MultiFileMemoryBuffer.**

Drop files?

React to uploads with server-side listeners:

- `StartedListener` - upload has started
- `SucceededListener` - a file has been uploaded successfully; implement at least this
- `AllFinishedListener` - all files of a multi-file upload have finished
- `FileRejectedListener` - file was rejected (for example - larger than Upload's `MaxFileSize`)
- etc.

Drop files?

Simple in-memory example:

```
MemoryBuffer memoryBuffer = new MemoryBuffer();
Upload upload = new Upload(memoryBuffer);

upload.addSucceededListener(event -> {
    // you can get MIME type, content length and the file name from the event object
    String text;
    try {
        text = IOUtils.toString(memoryBuffer.getInputStream(), StandardCharsets.UTF_8);
    } catch (IOException e) {
        text = "[exception reading stream]";
    }
    add(new TextArea("Text from uploaded file " + event.getFileName() + ":", text, ""));
});
```

Feedback

bit.ly/vaadin-training