How to detect a drop and do something with the dragged elements

In this example, which is a continuation of the previous example, we show how to drag an element and detect a drop, getting back a value that corresponds to the dragged element. Then we change the page content accordingly to the dropped element.

STEP 1: IN THE `DRAGSTART` HANDLER, COPY A VALUE IN THE DRAG AND DROP CLIPBOARD FOR LATER USE

In the `dragstart` handler, when a draggable `<li>` element has been dragged, get the value of its data-value attribute, and copy it to*the "drag and drop clipboard"* for later use.

When a value is copied to this clipboard, a key/name must be given. Data copied to the clipboard is associated with this name.

The variable `event.target` at *line 5* below is the `<li>` element that has been dragged, and event.target.dataset.value is the value of its data-value attribute (in our case apples, oranges or pears):

```
function dragStartHandler(event) {
    console.log('dragstart event, target:
' +event.target.innerHTML);
    // Copy to the drag'n'drop clipboard the value of the
    // data* attribute of the target,
    // with a type "Fruit".

  event.dataTransfer.setData("Fruit",event.target.dataset.value);
 }
```

STEP 2: DEFINE A "DROP ZONE"

Any visible HTML element may become a "drop zone"; just add an event listener for the `drop` event. Note that most of the time, as events may be propagated, we will also listen to `dragover` or`dragend` events and stop the propagation. More on this later...

```
<div ondragover="return false"ondrop="dropHandler(event);">
Drop your favorite fruits below:
 <ol id="droppedFruits"></ol>
</div>
```

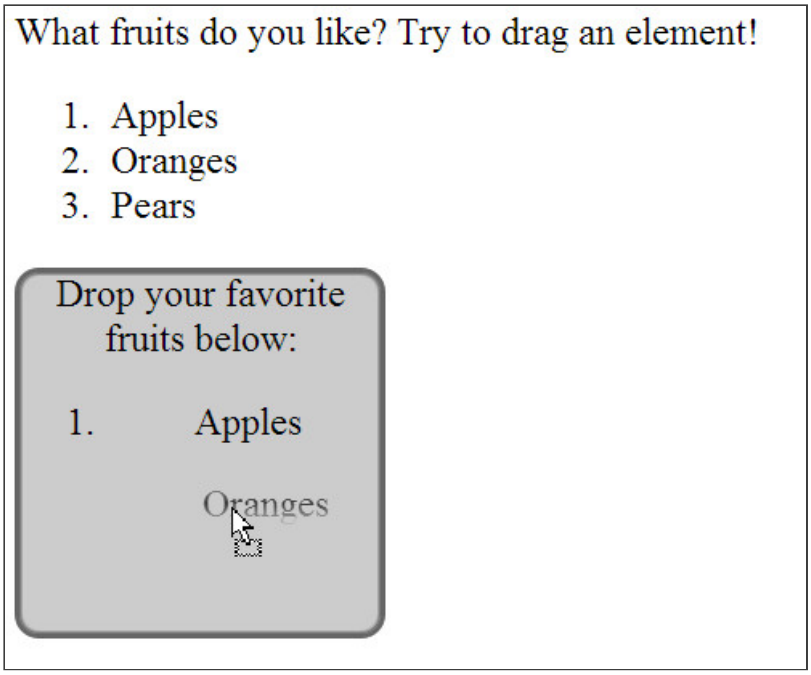
The `ondragover` handler will avoid propagating `dragover` events that may occur in high numbers while the mouse is moving above the drop zone while an element is being dragged. This is done by returning the `false` value at line 1.

STEP 3: WRITE A `DROP` HANDLER, GET CONTENT FROM THE CLIPBOARD, DO SOMETHING WITH IT

```
function dropHandler(event) {
   console.log('drop event, target:
' +event.target.innerHTML);
   ...
   // get the data from the drag'n'drop clipboard,
   // with a type="Fruit"
   var data =event.dataTransfer.getData("Fruit");
   // do something with the data
11.    ...
}
```

Typically, in the `drop` handler, once we acquired the data about the element that has been dropped (we get this from the clipboard at*line 7),* the data has been copied there during step 1 in the`dragstart` handler.

COMPLETE EXAMPLE



Try it in your browser below or play with it at CodePen:

HTML CSS Result Edit

What fruits do you like? Try to drag an element!

1. Apples
2. Oranges
3. Pears

Drop your favorite fruits below:

Source code:

```
<!DOCTYPE html>
<html>
 <head>
   <script>
```

```
        function dragStartHandler(event) {
            console.log('dragstart event, target: ' +
                        event.target.innerHTML);
            // Copy to the drag'n'drop clipboard the value
            // of the data* attribute of
            // the target, with a type "Fruits".
            event.dataTransfer.setData("Fruit",
                                       event.target.dataset.value);
        }
14.
        function dropHandler(event) {
           console.log('drop event, target: ' +
                                       event.target.innerHTML);
           var li =document.createElement('li');
           // get the data from the drag'n'drop clipboard,
           // with a type="Fruit"
           var data =event.dataTransfer.getData("Fruit");
           if (data == 'fruit-apple') {
              li.textContent = 'Apples';
           } else if (data == 'fruit-orange') {
26.           li.textContent = 'Oranges';
           } else if (data == 'fruit-pear') {
              li.textContent = 'Pears';
           } else {
              li.textContent = 'Unknown Fruit';
           }
           // add the dropped data as a child of the list.

 document.querySelector("#droppedFruits").appendChild(li);
        }
  </script>
36. </head>
<body>
   <p>What fruits do you like? Try to drag an element!</p>
   <ol ondragstart="dragStartHandler(event)">
      <li draggable="true" data-value="fruit-
apple">Apples</li>
      <li draggable="true" data-value="fruit-
orange">Oranges</li>
      <li draggable="true" data-value="fruit-pear">Pears</li>
```

```
    </ol>

<div ondragover="return false"ondrop="dropHandler(event);">
      Drop your favorite fruits below:
      <ol id="droppedFruits"></ol>
    </div>
<body>
<html>
```

In the above code, note:

- *Line 44*: we define the zone where we can drop (`ondrop=...`) and when a drag enters the zone, we stop the event propagation (`ondragover="return false"`)
- When we enter the `dragstart` listener (*line 5*), we copy the content of the `data-value` attribute of the object that is being dragged to the drag'n'drop clipboard with a name/key equal to "Fruit" (*line 11*),
- When a drop occurs in the "drop zone" (the `<div>` at *line 44*), the`dropHandler(event)` function is called. This always occurs after a call to the `dragstart` handler. In other words: when we enter the `drop` handler, there is always something in the clipboard! We do an `event.dataTransfer.setData(...)` in the `dragstart`handler, and an `event.dataTransfer.getData(...)` in the drop handler.
- The `dropHandler` function is called (*line 15*), we get the object (*line 21*) that is in the clipboard (the one with a name/key equal to "Fruit") , we create a `<li>` element (*line 18*) and we set its value depending on the value read in the clipboard (*lines 23-31*),
- Finally we add the `<li>` element to the `<ol>` list that is in the`<div>`.

Notice that we use some CSS to improve the drop zone (not presented in the source code above, but available in the online example):

```
    div {
       height: 150px;
       width: 150px;
       float: left;
```

```
    border: 2px solid #666666;
    background-color: #ccc;
    margin-right: 5px;
    border-radius: 10px;
    box-shadow: inset 0 0 3px #000;
    text-align: center;
    cursor: move;
 }
li:hover {
    border: 2px dashed #000;
}
```