Event Handling: HTML Drag-And-Drop API

Dragging and dropping a page element is a fun and convenient way for users to interact with a Web page! HTML drag-and-drop interfaces are most commonly used for dragging files, such as PDFs or images, onto a page in a specified area known as a *drop zone*.

While less typical than button clicks and form submission events, HTML drag-and-drop is relevant to event handling because it uses the DOM event modeland drag eventsinherited from mouse events.

This reading will go over how to use the HTML Drag and Drop API to create "draggable" page elements and allow users to drop them into a drop zone.

Basic drag-and-drop functions

Let's go over how to set up basic drag-and-drop functionality, according to theMDN documentation. You'll need to mark an element as "draggable". Then, to do something with that dragging, you need

- A *dragstart*handler -- occurs when the user clicks the mouse and starts dragging
- A *drop*handler -- occurs when the user releases the mouse click and "drops" the element

You will also see how to use the `dragenter`and `dragleave`events to do some nice interactions.

The example

Here's an HTML document that you can copy and paste into a text editor if you want to follow along.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Red Square is a Drag</title>
  <style>
    #red-square {
      background-color: red;
      box-sizing: border-box;
      height: 100px;
      width: 100px;
    }
  </style>
</head>
<body>
  <div id="red-square"></div>
</body>
</html>
```

You don't need to know what `box-sizing` does. That'll be covered in future lessons. It's in this example to make sure the box looks ok when it's dragged.

The first step of drag and drop

The first step to making an element draggable is to add that attribute to the element itself. Change the red square `div` to have the `draggable` attribute with a value of "true".

```
<div id="red-square" draggable="true"></div>
```

Now, if you refresh your page, you can start dragging the red square. When you release it, it will "snap" back to it's original position.

Handling the start of a drag

Now that the element is draggable, you need some JavaScript to handle the event of when someone starts dragging an element. This is there so that your code knows what's being dragged! Otherwise, how will it know what to do when the dragging ends?

The following code creates a handler for the `dragstart` event. Then, it subscribes to the red square's `dragstart` event with that event handler. The event handler, in this case, will add a class to the red square to make it show that it's being dragged. Then, it adds the value of the `id` of the element to the

"data transfer" object. The "data transfer" object holds all of the information that will be needed when the dragging operation ends.

The `classList` object for the HTML element is just a way to add and remove CSS classes to and from a DOM object. You're not going to be tested over that bit of information for some weeks, so don't worry about remembering it. Just understand that using the `add` method on it *adds* the CSS class to the HTML element.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Red Square is a Drag</title>
  <script type="text/javascript">
    const handleDragStart = e => {
      e.target.classList.add('is-being-dragged');
      e.dataTransfer.setData('text/plain', e.target.id);
      e.dataTransfer.dropEffect = 'move';
    }
    window.addEventListener('DOMContentLoaded', () => {
      document
        .getElementById('red-square')
        .addEventListener('dragstart', handleDragStart);
    });
  </script>
  <style>
    #red-square {
      background-color: red;
      box-sizing: border-box;
      height: 100px;
      width: 100px;
    }

    .is-being-dragged {
      opacity: 0.5;
      border: 2px dashed white;
```

```
    }
  </style>
</head>
<body>
  <div id="red-square"></div>
</body>
</html>
```

If you update your version of the code, you can now see that the square's border gets all dashy when you drag it!

The drop zone is all clear!

A drop zone is just another HTML element. Put another `div` in your HTML and give it an id of `drop-zone`. It could, conceivably, look like this.

```
<body>
  <div id="red-square"></div>
```

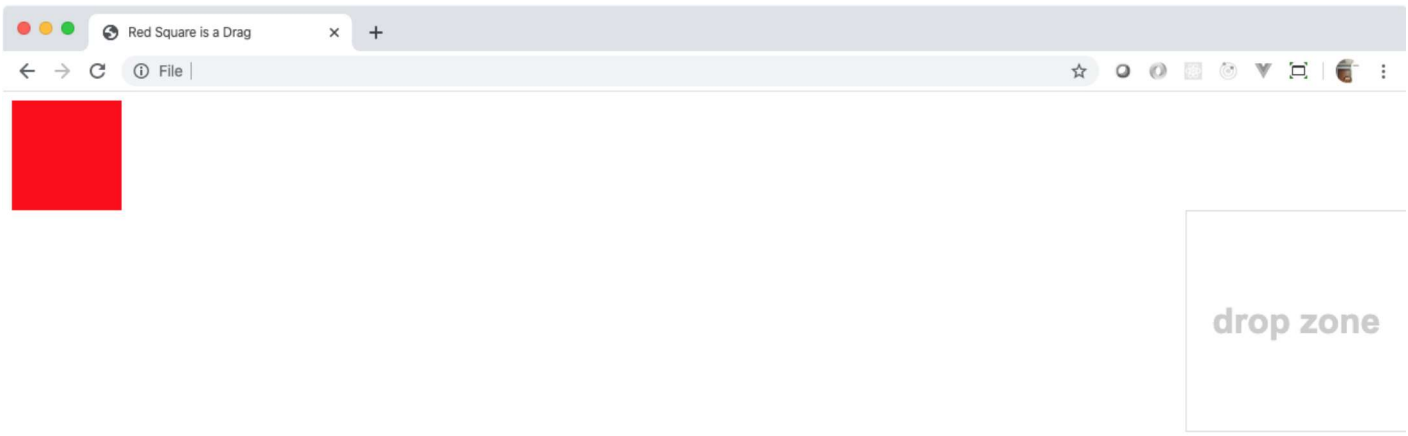
```
  <div id="drop-zone">drop zone</div>
</body>
```

To make it visible, add some CSS. Note that most of that in there is to make it look pretty. You should understand the simple things like "background-color" and "color" and "font-size", "height" and "width". You won't yet be tested on any of those other properties. But, feel free to play around with them!

```
<style>
  #drop-zone {
    align-items: center;
    border: 1px solid #DDD;
    color: #CCC;
    display: flex;
    font-family: Arial, Helvetica, sans-serif;
    font-size: 2em;
    font-weight: bold;
    height: 200px;
    justify-content: center;
    position: absolute;
    right: 0;
    width: 200px;
  }

  #red-square {
    background-color: red;
    box-sizing: border-box;
    height: 100px;
    width: 100px;
  }

  .is-being-dragged {
    opacity: 0.5;
    border: 2px dashed white;
  }
</style>
```



Handle when the red square gets enters the drop zone (and leaves)

This is another couple of JavaScript event handlers, but this time, it will handle the `dragenter` and `dragleave` events on the drop zone element. You can replace the `<script>` tag in your HTML with this version, here, to handle those events. Note that the `handleDragEnter` event handler merely adds a CSS class to the drop zone. The `handleDragLeave` removes the CSS class. Then, in the `DOMContentLoaded` event handler, the last three lines gets a reference to the drop zone element and attaches the event handlers to it.

```
<script type="text/javascript">
  const handleDragStart = e => {
    e.target.classList.add('is-being-dragged');
    e.dataTransfer.setData('text/plain', e.target.id);
    e.dataTransfer.dropEffect = 'move';
  };
```

```
  const handleDragEnter = e => {
    e.target.classList.add('is-active-drop-zone');
  };

  const handleDragLeave = e => {
    e.target.classList.remove('is-active-drop-zone');
  };

  window.addEventListener('DOMContentLoaded', () => {
    document
      .getElementById('red-square')
      .addEventListener('dragstart', handleDragStart);

    const dropZone = document.getElementById('drop-zone');
    dropZone.addEventListener('dragenter', handleDragEnter);
    dropZone.addEventListener('dragleave', handleDragLeave);
  });
</script>
```

The CSS to make the item change looks like this. Just add the class to the `<style>` tag in the HTML.

```
.is-active-drop-zone {
  background-color: blue;
}
```

Just look at that drop zone turn blue with glee!

Do something with a "drop"!

Finally, the `drop` event of the drop target handles what happens when you let go of the draggable element over the drop zone. However, there's one small problem. From the MDN documentation on Drag Operations:

If the mouse is released over an element that is a valid drop target, that is, one that cancelled the last `dragenter` or `dragover` event, then the drop will be successful, and a drop event will fire at the target. Otherwise, the drag operation is cancelled, and no drop event is fired.

For this to work properly, you will *also* have to subscribe to the `drop` event for the drop zone. Then, in both the handlers for the `drop` and `dragenter` events, you'll need to cancel the event. Recall that in the last article you learned about the `preventDefault()` method on the event object. That's what you need to call in both the `drop` and `dragenter` event handlers to make the `drop` event fire.

That's a lot of words! Here is the final HTML for this little dragging example.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Red Square is a Drag</title>
  <script type="text/javascript">
    const handleDragStart = e => {
      e.target.classList.add('is-being-dragged');
      e.dataTransfer.setData('text/plain', e.target.id);
      e.dataTransfer.dropEffect = 'move';
    };

    const handleDragEnter = e => {
      // Needed so that the "drop" event will fire.
      e.preventDefault();
      e.target.classList.add('is-active-drop-zone');
    };

    const handleDragLeave = e => {
      e.target.classList.remove('is-active-drop-zone');
    };

    const handleDragOver = e => {
      // Needed so that the "drop" event will fire.
      e.preventDefault();
    };

    const handleDrop = e => {
      const id = e.dataTransfer.getData('text/plain');
      const draggedElement = document.getElementById(id);
      draggedElement.draggable = false;
      e.target.appendChild(draggedElement);
    };

    window.addEventListener('DOMContentLoaded', () => {
      document
        .getElementById('red-square')
```

```
      .addEventListener('dragstart', handleDragStart);

    const dropZone = document.getElementById('drop-zone');
    dropZone.addEventListener('drop', handleDrop);
    dropZone.addEventListener('dragenter', handleDragEnter);
    dropZone.addEventListener('dragleave', handleDragLeave);
    dropZone.addEventListener('dragover', handleDragOver);
  });
</script>
<style>
  #drop-zone {
    align-items: center;
    border: 1px solid #DDD;
    color: #CCC;
    display: flex;
    font-family: Arial, Helvetica, sans-serif;
    font-size: 2em;
    font-weight: bold;
    height: 200px;
    justify-content: center;
    position: absolute;
    right: 0;
    top: 100px;
    width: 200px;
  }

  #red-square {
    background-color: red;
    box-sizing: border-box;
    height: 100px;
    width: 100px;
  }

  .is-being-dragged {
    opacity: 0.5;
    border: 8px dashed white;
  }

  .is-active-drop-zone {
    background-color: blue;
```

```
      color:
    }
  </style>
  </style>
</head>
<body>
  <div id="red-square" draggable="true"></div>
  <div id="drop-zone">drop zone</div>
</body>
</html>
```

What you learned:

There is *a lot*going on, here. Here's a quick review of what you should get from this article.

- HTML has a Drag and Drop API that you can use to do dragging and dropping in an application.

- To make it so that a person can drag an HTML element around, you add the `draggable="true"` attribute/value pair to the element (or elements) that you want to drag.
- You can subscribe to one event on the thing you want to drag, the `dragstart` event. That event will allow you to customize the element and save data to the `dataTransfer` object.
- You can subscribe to four events for the "drop zone" element(s): `dragenter`, `dragover`, and `dragleave`. If you want the `drop` event to work, you *must* subscribe to both `dragenter` and `dragover` and cancel the event using the `preventDefault()` method of the event.