HTML Drag and Drop API

HTML Drag and Drop interfaces enable applications to use drag-and-drop features in browsers. The user may select *draggable* elements with a mouse, drag those elements to a *droppable* element, and drop them by releasing the mouse button. A translucent representation of the *draggable* elements follows the pointer during the drag operation.

For web sites, extensions, and XUL applications, you can customize which elements can become *draggable*, the type of feedback the *draggable* elements produce, and the *droppable* elements.

This overview of HTML Drag and Drop includes a description of the interfaces, basic steps to add drag-and-drop support to an application, and an interoperability summary of the interfaces.

Drag Events

HTML drag-and-drop uses the `DOM event model` and *`drag events`* inherited from `mouse events`. A typical *drag operation* begins when a user selects a *draggable* element, drags the element to a *droppable* element, and then releases the dragged element.

During drag operations, several event types are fired, and some events might fire many times, such as the `drag` and `dragover` events.

Each drag event type has an associated global event handler:

Event	On Event Handler	Fires when...
`drag`	`ondrag`	…a *dragged item* (element or text selection) is dragged.
`dragend`	`ondragend`	…a drag operation ends (such as releasing a mouse button or hitting the Esc key; see Finishing a Drag.)

Event	On Event Handler	Fires when...
`dragenter`	`ondragenter`	…a dragged item enters a valid drop target. (See Specifying Drop Targets.)
`dragexit`	`ondragexit`	…an element is no longer the drag operation's immediate selection target.
`dragleave`	`ondragleave`	…a dragged item leaves a valid drop target.
`dragover`	`ondragover`	…a dragged item is being dragged over a valid drop target, every few hundred milliseconds.
`dragstart`	`ondragstart`	…the user starts dragging an item. (See Starting a Drag Operation.)
`drop`	`ondrop`	…an item is dropped on a valid drop target. (See Performing a Drop.)

> **Note:** Neither `dragstart` nor `dragend` events are fired when dragging a file into the browser from the OS.

Interfaces

The HTML Drag and Drop interfaces are `DragEvent`, `DataTransfer`, `DataTransferItem` and `DataTransferItemList`.

The `DragEvent` interface has a constructor and one `dataTransfer` property, which is a `DataTransfer` object.

`DataTransfer` objects include the drag event's state, such as the type of drag being done (like `copy` or `move`), the drag's data (one or more items), and the MIME type of each *drag item*. `DataTransfer` objects also have methods to add or remove items to the drag's data.

The `DragEvent` and `DataTransfer` interfaces should be the only ones needed to add HTML Drag and Drop capabilities to an application. (Firefox supports some Gecko-specific extensions to the `DataTransfer` object, but those extensions will only work on Firefox.)

Each `DataTransfer` object contains an `items` property, which is a `list` of `DataTransferItem` objects. A `DataTransferItem` object represents a single *drag item*, each with a `kind` property (either `string` or `file`) and a `type` property for the data item's MIME type. The `DataTransferItem` object also has methods to get the drag item's data.

The `DataTransferItemList` object is a list of `DataTransferItem` objects. The list object has methods to add a drag item to the list, remove a drag item from the list, and clear the list of all drag items.

A key difference between the `DataTransfer` and `DataTransferItem` interfaces is that the former uses the synchronous `getData()` method to access a drag item's data, but the latter instead uses the asynchronous `getAsString()` method.

> **Note:** `DragEvent` and `DataTransfer` are broadly supported on desktop browsers. However, the `DataTransferItem` and `DataTransferItemList` interfaces have limited browser support. See Interoperability for more information about drag-and-drop interoperability.

Gecko-specific interfaces

Mozilla and Firefox support some features not in the standard drag-and-drop model. These are *convenience functions* to help with dragging multiple items or non-string data (such as files). For more information, see Dragging and Dropping Multiple Items. Additionally, see the `DataTransfer` reference page for all of the Gecko-specific properties and Gecko-specific methods.

The basics

This section is a summary of the basic steps to add drag-and-drop functionality to an application.

Identify what is *draggable*

Making an element *draggable* requires adding the `draggable` attribute and the `ondragstart` global event handler, as shown in the following code sample:

```
<script>
  function dragstart_handler(ev) {
    // Add the target element's id to the data transfer object
    ev.dataTransfer.setData("text/plain", ev.target.id);
  }

  window.addEventListener('DOMContentLoaded', () => {
    // Get the element by id
    const element = document.getElementById("p1");
    // Add the ondragstart event listener
    element.addEventListener("dragstart", dragstart_handler);
  });
</script>

<p id="p1" draggable="true">This element is draggable.</p>
```

For more information, see:

- Draggable attribute reference
- Drag operations guide

Define the drag's data

The application is free to include any number of data items in a drag operation. Each data item is a `string` of a particular `type` — typically a MIME type such as `text/html`.

Each `drag event` has a `dataTransfer` property that *holds* the event's data. This property (which is a `DataTransfer` object) also has methods to *manage* drag data. The `setData()` method is used to add an item to the drag data, as shown in the following example.

```
function dragstart_handler(ev) {
  // Add different types of drag data
  ev.dataTransfer.setData("text/plain", ev.target.innerText);
  ev.dataTransfer.setData("text/html", ev.target.outerHTML);
  ev.dataTransfer.setData("text/uri-list", ev.target.ownerDocument.loc
}
```

- For a list of common data types used in drag-and-drop (such as text, HTML, links, and files), see Recommended Drag Types.

- For more information about drag data, see Drag Data.

Define the drag image

By default, the browser supplies an image that appears beside the pointer during a drag operation. However, an application may define a custom image with the `setDragImage()` method, as shown in the following example.

```
function dragstart_handler(ev) {
  // Create an image and then use it for the drag image.
  // NOTE: change "example.gif" to a real image URL or the image
  // will not be created and the default drag image will be used.
  let img = new Image();
  img.src = 'example.gif';
  ev.dataTransfer.setDragImage(img, 10, 10);
}
```

Learn more about drag feedback images in:

- Setting the Drag Feedback Image

Define the drag *effect*

The `dropEffect` property is used to control the feedback the user is given during a drag-and-drop operation. It typically affects which cursor the browser displays while dragging. For example, when the user hovers over a drop target, the browser's cursor may indicate the type of operation that will occur.

Three effects may be defined:

1. **`copy`** indicates that the dragged data will be copied from its present location to the drop location.
2. **`move`** indicates that the dragged data will be moved from its present location to the drop location.
3. **`link`** indicates that some form of relationship or connection will be created between the source and drop locations.

During the drag operation, drag effects may be modified to indicate that certain effects are allowed at certain locations.

The following example shows how to use this property.

```
function dragstart_handler(ev) {
  ev.dataTransfer.dropEffect = "copy";
}
```

For more details, see:

- Drag Effects

Define a *drop zone*

By default, the browser prevents anything from happening when dropping something onto most HTML elements. To change that behavior so that an element becomes a *drop zone* or is *droppable*, the element must have both `ondragover` and `ondrop` event handler attributes.

The following example shows how to use those attributes, and includes basic event handlers for each attribute.

```
<script>
function dragover_handler(ev) {
 ev.preventDefault();
 ev.dataTransfer.dropEffect = "move";
}
function drop_handler(ev) {
 ev.preventDefault();
 // Get the id of the target and add the moved element to the target's
 const data = ev.dataTransfer.getData("text/plain");
 ev.target.appendChild(document.getElementById(data));
}
</script>

<p id="target" ondrop="drop_handler(event)" ondragover="dragover_handl
```

Note that each handler calls `preventDefault()` to prevent additional event processing for this event (such as touch events or pointer events).

For more information, see:

- Specifying Drop Targets

Handle the drop *effect*

The handler for the `drop` event is free to process the drag data in an application-specific way.

Typically, an application uses the `getData()` method to retrieve drag items and then process them accordingly. Additionally, application semantics may differ depending on the value of the `dropEffect` and/or the state of modifier keys.

The following example shows a drop handler getting the source element's `id` from the drag data, and then using the `id` to move the source element to the drop element:

```
<script>
function dragstart_handler(ev) {
 // Add the target element's id to the data transfer object
 ev.dataTransfer.setData("application/my-app", ev.target.id);
 ev.dataTransfer.dropEffect = "move";
}
function dragover_handler(ev) {
 ev.preventDefault();
 ev.dataTransfer.dropEffect = "move"
}
function drop_handler(ev) {
 ev.preventDefault();
 // Get the id of the target and add the moved element to the target's
 const data = ev.dataTransfer.getData("application/my-app");
 ev.target.appendChild(document.getElementById(data));
}
</script>

<p id="p1" draggable="true" ondragstart="dragstart_handler(event)">Thi
<div id="target" ondrop="drop_handler(event)" ondragover="dragover_han
```

For more information, see:

- Performing a Drop

Drag end

At the end of a drag operation, the `dragend` event fires at the *source* element — the element that was the target of the drag start.

This event fires regardless of whether the drag completed or was canceled. The `dragend` event handler can check the value of the `dropEffect` property to determine if the drag operation succeeded or not.

For more information about handling the end of a drag operation, see:

- Finishing a Drag

Interoperability

As can be seen in the DataTransferItem interface's Browser Compatibility table, drag-and-drop interoperability is relatively broad among desktop browsers (except the `DataTransferItem` and `DataTransferItemList` interfaces have less support). This data also indicates drag-and-drop support among mobile browsers is very low.

Examples and demos

- Copying and moving elements with the `DataTransfer` interface
- Copying and moving elements with the `DataTransferListItem` interface
- Dragging and dropping files (Firefox only): http://jsfiddle.net/9C2EF/
- Dragging and dropping files (All browsers): https://jsbin.com/hiqasek/
- A parking project using the Drag and Drop API: https://park.glitch.me/ (You can edit here)

Specifications

Specification	Status	Comment
HTML Living Standard	LS Living Standard	

See also

- Drag Operations
- Dragging and Dropping Multiple Items
- Recommended Drag Types
- HTML5 Living Standard: Drag and Drop
- Drag and Drop interoperability data from CanIUse

Last modified: Jun 22, 2020, by MDN contributors

Related Topics

HTML Drag and Drop API

▼ Guides

File drag and drop

Drag Operations

Dragging and Dropping Multiple Items

Recommended Drag Types

▼ Interfaces

`DataTransfer`

`DataTransferItem`

`DataTransferItemList`

`DragEvent`

▼ Events

`HTMLElement: drag`

```
HTMLElement: drag
HTMLElement: dragend
HTMLElement: dragenter
HTMLElement: dragexit
HTMLElement: dragleave
HTMLElement: dragover
HTMLElement: dragstart
HTMLElement: drop
```