Audio mixer dev notes

Source:
https://developer.mozilla.org/en-US/docs/Web/API/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.)
`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.)

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.

<audio>: The Embed Audio element

Source: https://developer.mozilla.org/en-US/docs/Web/API/HTMLAudioElement

The **HTML `<audio>` element** is used to embed sound content in documents. It may contain one or more audio sources, represented using the `src` attribute or the `<source>` element: the browser will choose the most suitable one. It can also be the destination for streamed media, using a `MediaStream`.

Events

Event name	Fired when
`audioprocess`	The input buffer of a `ScriptProcessorNode` is ready to be processed.
`canplay`	The browser can play the media, but estimates that not enough data has been loaded to play the media up to its end without having to stop for further buffering of content.

`canplaythrough`	The browser estimates it can play the media up to its end without stopping for content buffering.
`complete`	The rendering of an `OfflineAudioContext` is terminated.
`durationchange`	The `duration` attribute has been updated.
`emptied`	The media has become empty; for example, this event is sent if the media has already been loaded (or partially loaded), and the `load()` method is called to reload it.
`ended`	Playback has stopped because the end of the media was reached.
`loadeddata`	The first frame of the media has finished loading.
`loadedmetadata`	The metadata has been loaded.
`pause`	Playback has been paused.
`play`	Playback has begun.
`playing`	Playback is ready to start after having been paused or delayed due to lack of data.
`ratechange`	The playback rate has changed.
`seeked`	A *seek* operation completed.

`seeking`	A *seek* operation began.
`stalled`	The user agent is trying to fetch media data, but data is unexpectedly not forthcoming.
`suspend`	Media data loading has been suspended.
`timeupdate`	The time indicated by the `currentTime` attribute has been updated.
`volumechange`	The volume has changed.
`waiting`	Playback has stopped because of a temporary lack of data