HTML5 - DRAG & DROP

Drag and Drop *DnD* is powerful User Interface concept which makes it easy to copy, reorder and deletion of items with the help of mouse clicks. This allows the user to click and hold the mouse button down over an element, drag it to another location, and release the mouse button to drop the element there.

To achieve drag and drop functionality with traditional HTML4, developers would either have to either have to use complex Javascript programming or other Javascript frameworks like jQuery etc.

Now HTML 5 came up with a Drag and Drop *DnD* API that brings native DnD support to the browser making it much easier to code up.

HTML 5 DnD is supported by all the major browsers like Chrome, Firefox 3.5 and Safari 4 etc.

Drag and Drop Events

There are number of events which are fired during various stages of the drag and drop operation. These events are listed below −

Events	Description
dragstart	Fires when the user starts dragging of the object.
dragenter	Fired when the mouse is first moved over the target element while a drag is occuring. A listener for this event should indicate whether a drop is allowed over this location. If there are no listeners, or the listeners perform no operations, then a drop is not allowed by default.
dragover	This event is fired as the mouse is moved over an element when a drag is occuring. Much of the time, the operation that occurs during a listener will be the same as the dragenter event.
dragleave	This event is fired when the mouse leaves an element while a drag is occuring. Listeners should remove any highlighting or insertion markers used for drop feedback.
drag	Fires every time the mouse is moved while the object is being dragged.
drop	The drop event is fired on the element where the drop was occured at the end of the drag operation. A listener would be responsible for retrieving the data being dragged and inserting it at the drop location.
dragend	Fires when the user releases the mouse button while dragging an object.

Note − Note that only drag events are fired; mouse events such as *mousemove* are not fired during a drag operation.

The DataTransfer Object

The event listener methods for all the drag and drop events accept **Event** object which has a readonly attribute called **dataTransfer**. The **event.dataTransfer** returns **DataTransfer** object associated with the event as follows −

```
function EnterHandler(event) {
   DataTransfer dt = event.dataTransfer;
   ..............
}
```

The *DataTransfer* object holds data about the drag and drop operation. This data can be retrieved

and set in terms of various attributes associated with DataTransfer object as explained below:

Sr.No.	DataTransfer attributes and their description
1	**dataTransfer.dropEffect [= value]** • Returns the kind of operation that is currently selected. • This attribute can be set, to change the selected operation. • The possible values are **none, copy, link,** and **move**.
2	**dataTransfer.effectAllowed [= value]** • Returns the kinds of operations that are to be allowed. • This attribute can be set, to change the allowed operations. • The possible values are **none, copy, copyLink, copyMove, link, linkMove, move, all** and **uninitialized**.
3	**dataTransfer.types** Returns a DOMStringList listing the formats that were set in the dragstart event. In addition, if any files are being dragged, then one of the types will be the string "Files".
4	**dataTransfer.clearData[*format*]** Removes the data of the specified formats. Removes all data if the argument is omitted.
5	**dataTransfer.setData*format, data*** Adds the specified data.
6	**data = dataTransfer.getData*format*** Returns the specified data. If there is no such data, returns the empty string.
7	**dataTransfer.files** Returns a FileList of the files being dragged, if any.
8	**dataTransfer.setDragImage*element, x, y*** Uses the given element to update the drag feedback, replacing any previously specified feedback.
9	**dataTransfer.addElement*element*** Adds the given element to the list of elements used to render the drag feedback.

Drag and Drop Process

Following are the steps to be carried out to implement Drag and Drop operation −

Step 1: Making an Object Draggable

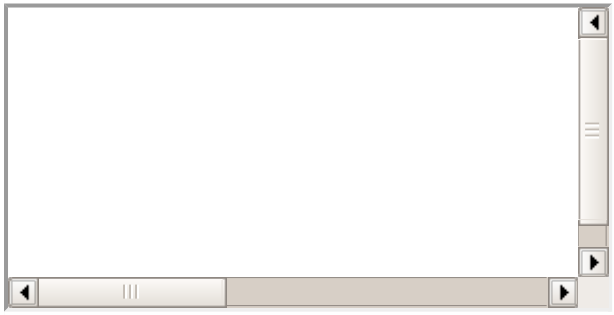
Here are steps to be taken −

- If you want to drag an element, you need to set the **draggable** attribute to **true** for that element.
- Set an event listener for **dragstart** that stores the data being dragged.
- The event listener **dragstart** will set the allowed effects *copy*, *move*, *link*, *orsomecombination*.

Following is the example to make an object dragable −

```
<!DOCTYPE HTML>
<html>
   <head>

      <style type="text/css">
         #boxA, #boxB {
            float:left;padding:10px;margin:10px; -moz-user-select:none;
         }

         #boxA { background-color: #6633FF; width:75px; height:75px;  }
         #boxB { background-color: #FF6699; width:150px; height:150px; }
      </style>

      <script type="text/javascript">
         function dragStart(ev) {
            ev.dataTransfer.effectAllowed='move';
            ev.dataTransfer.setData("Text", ev.target.getAttribute('id'));
            ev.dataTransfer.setDragImage(ev.target,0,0);

            return true;
         }
      </script>

   </head>
   <body>

      <center>
         <h2>Drag and drop HTML5 demo</h2>
         <div>Try to drag the purple box around.</div>

         <div
            ondragstart="return dragStart(event)">
            <p>Drag Me</p>
         </div>

         <div >Dustbin</div>
      </center>

   </body>
</html>
```

This will produce following result −



Step 2: Dropping the Object

To accept a drop, the drop target has to listen to at least three events.

- The **dragenter** event, which is used to determine whether or not the drop target is to accept the drop. If the drop is to be accepted, then this event has to be canceled.
- The **dragover** event, which is used to determine what feedback is to be shown to the user. If the event is canceled, then the feedback *typicallythecursor* is updated based on the dropEffect attribute's value.
- Finally, the **drop** event, which allows the actual drop to be performed.

Following is the example to drop an object into another object −

```
<!DOCTYPE HTML>
<html>
   <head>

      <style type="text/css">
         #boxA, #boxB {
            float:left;padding:10px;margin:10px;-moz-user-select:none;
         }

         #boxA { background-color: #6633FF; width:75px; height:75px;  }
         #boxB { background-color: #FF6699; width:150px; height:150px; }
      </style>

      <script type="text/javascript">
         function dragStart(ev) {
            ev.dataTransfer.effectAllowed='move';
            ev.dataTransfer.setData("Text", ev.target.getAttribute('id'));
            ev.dataTransfer.setDragImage(ev.target,0,0);

            return true;
         }

         function dragEnter(ev) {
            event.preventDefault();
            return true;
         }

         function dragOver(ev) {
            return false;
         }

         function dragDrop(ev) {
            var src = ev.dataTransfer.getData("Text");
            ev.target.appendChild(document.getElementById(src));
            ev.stopPropagation();
            return false;
         }
      </script>

   </head>
   <body>

      <center>
         <h2>Drag and drop HTML5 demo</h2>
         <div>Try to move the purple box into the pink box.</div>

         <div
            ondragstart="return dragStart(event)">
            <p>Drag Me</p>
         </div>

         <div
            ondrop="return dragDrop(event)"
            ondragover="return dragOver(event)">Dustbin
         </div>
```

```
      </center>

   </body>
</html>
```

This will produce following result −