**Introduction**

For years, we've been using libraries like JQuery and Dojo to simplify complex UI elements like animations, rounded corners, and drag and drop. There's no doubt, eye-candy is important for making rich, immersive experiences on the web. But why should a library be required for common tasks that all developers are using?

[Drag and drop](http://www.whatwg.org/specs/web-apps/current-work/multipage/dnd.html#dnd) (DnD) is a first class citizen in HTML5! The spec defines an event-based mechanism, JavaScript API, and additional markup for declaring that just about any type of element be draggable on a page. I don't think anyone can argue against native browser support for a particular feature. Native browser DnD means faster, more responsive web apps.

**Feature Detection**

Many apps that utilize DnD would have a poor experience without it. For example, imagine a chess game in which the pieces don't move. Oops! Although browser support is fairly complete, determining if a browser implements DnD (or any HTML5 feature for that matter) is important for providing a solution that degrades nicely. When/where DnD isn't available, fire up that library fallback to maintain a working app.

If you need to rely on an API, always use feature detection rather than sniffing the browser's User-Agent. One of the better libraries for feature detection is [Modernizr](http://www.modernizr.com/). Modernizr sets a boolean property for each feature it tests. Thus, checking for DnD is a one-liner:

if (Modernizr.draganddrop) {

*// Browser supports HTML5 DnD.*

} else {

*// Fallback to a library solution.*

}

**Creating draggable content**

Making an object draggable is simple. Set the draggable=true attribute on the element you want to make moveable. Just about anything can be drag-enabled, including images, links, files, or other DOM nodes.

As an example, let's start creating rearrangeable columns. The basic markup may look something like this:

<div id="columns">

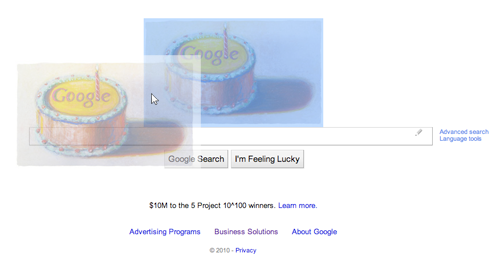
<div class="column" draggable="true"><header>A</header></div>

<div class="column" draggable="true"><header>B</header></div>

<div class="column" draggable="true"><header>C</header></div>

</div>

It's worth noting that in most browsers, text selections, img elements, and anchor elements with an href attribute are draggable by default. For example, dragging the logo on google.com produces a ghost image:

Most browsers support dragging an image by default.

which can be dropped in the address bar, a <input type="file" /> element, or even the desktop. If you want to enable other types of content to be draggable, you'll need to leverage the HTML5 DnD APIs.

Using a little CSS3 magic we can spruce up our markup to look like columns. Adding cursor: move gives users a visual indicator that something is moveable:

<style>

*/\* Prevent the text contents of draggable elements from being selectable. \*/*

[draggable] {

-moz-user-select: none;

-khtml-user-select: none;

-webkit-user-select: none;

user-select: none;

*/\* Required to make elements draggable in old WebKit \*/*

-khtml-user-drag: element;

-webkit-user-drag: element;

}

.column {

height: 150px;

width: 150px;

float: left;

border: 2px solid #666666;

background-color: #ccc;

margin-right: 5px;

-webkit-border-radius: 10px;

-ms-border-radius: 10px;

-moz-border-radius: 10px;

border-radius: 10px;

-webkit-box-shadow: inset 0 0 3px #000;

-ms-box-shadow: inset 0 0 3px #000;

box-shadow: inset 0 0 3px #000;

text-align: center;

cursor: move;

}

.column header {

color: #fff;

text-shadow: #000 0 1px;

box-shadow: 5px;

padding: 5px;

background: -moz-linear-gradient(left center, rgb(0,0,0), rgb(79,79,79), rgb(21,21,21));

background: -webkit-gradient(linear, left top, right top,

color-stop(0, rgb(0,0,0)),

color-stop(0.50, rgb(79,79,79)),

color-stop(1, rgb(21,21,21)));

background: -webkit-linear-gradient(left center, rgb(0,0,0), rgb(79,79,79), rgb(21,21,21));

background: -ms-linear-gradient(left center, rgb(0,0,0), rgb(79,79,79), rgb(21,21,21));

border-bottom: 1px solid #ddd;

-webkit-border-top-left-radius: 10px;

-moz-border-radius-topleft: 10px;

-ms-border-radius-topleft: 10px;

border-top-left-radius: 10px;

-webkit-border-top-right-radius: 10px;

-ms-border-top-right-radius: 10px;

-moz-border-radius-topright: 10px;

border-top-right-radius: 10px;

}

</style>

Result (draggable but won't do anything):

A

B

C

In the example above, most browsers will create a ghost image of the content being dragged. Others (FF in particular) will require that some [data be sent](https://html5rocks.appspot.com/en/tutorials/dnd/basics/#toc-dataTransfer) in the drag operation. In the next section, we'll start to make our column example more interesting by adding listeners to process the drag/drop event model.

**Listening for Dragging Events**

There are a number of different events to attach to for monitoring the entire drag and drop process:

* dragstart
* drag
* dragenter
* dragleave
* dragover
* drop
* dragend

To handle the DnD flow, we need the notion of a source element (where the drag originates), the data payload (what we're trying to drop), and a target (an area to catch the drop). The source element can be an image, list, link, file object, block of HTML...you name it. The target is the drop zone (or set of drop zones) that accepts the data the user is trying to drop. Keep in mind that not all elements can be targets (e.g. images).

**1. Starting a Drag**

Once you have draggable="true" attributes defined on your content, attachdragstart event handlers to kick off the DnD sequence for each column.

This code will set the column's opacity to 40% when the user begins dragging it:

function handleDragStart(e) {

this.style.opacity = '0.4'; *// this / e.target is the source node.*

}

var cols = document.querySelectorAll('#columns .column');

[].forEach.call(cols, function(col) {

col.addEventListener('dragstart', handleDragStart, false);

});

Result:

A

B

C

Since the dragstart event's target is our source element, setting this.style.opacity to 40% gives the user visual feedback that the element is the current selection being moved. One thing that we still need to do is return the columns opacity to 100% once the drag is done. An obvious place to handle that is the dragend event. More on this later.

**2. dragenter, dragover, and dragleave**

dragenter, dragover, and dragleave event handlers can be used to provide additional visual cues during the drag process. For example, when a column is hovered over during a drag, its border could become dashed. This will let users know the columns are also drop targets.

<style>

.column.over {

border: 2px dashed #000;

}

</style>

function handleDragStart(e) {

this.style.opacity = '0.4'; *// this / e.target is the source node.*

}

function handleDragOver(e) {

if (e.preventDefault) {

e.preventDefault(); *// Necessary. Allows us to drop.*

}

e.dataTransfer.dropEffect = 'move'; *// See the section on the DataTransfer object.*

return false;

}

function handleDragEnter(e) {

*// this / e.target is the current hover target.*

this.classList.add('over');

}

function handleDragLeave(e) {

this.classList.remove('over'); *// this / e.target is previous target element.*

}

var cols = document.querySelectorAll('#columns .column');

[].forEach.call(cols, function(col) {

col.addEventListener('dragstart', handleDragStart, false);

col.addEventListener('dragenter', handleDragEnter, false);

col.addEventListener('dragover', handleDragOver, false);

col.addEventListener('dragleave', handleDragLeave, false);

});

There are a couple of points worth covering in this code:

* The this/e.target changes for each type of event, depending on where we are in the DnD event model.
* In the case of dragging something like a link, we need to prevent the browser's default behavior, which is to navigate to that link. To do this, call e.preventDefault() in the dragover event. Another good practice is to return false in that same handler. Browsers are somewhat inconsistent about needing these, but they don't hurt to add.
* dragenter is used to toggle the 'over' class instead of the dragover. If we were to use dragover, our CSS class would be toggled many times as the event dragover continued to fire on a column hover. Ultimately, that would cause the browser's renderer to do a large amount of unnecessary work. Keeping redraws to a minimum is always a good idea.

**3. Completing a Drag**

To process the actual drop, add an event listener for the drop and dragend events. In this handler, you'll need to prevent the browser's default behavior for drops, which is typically some sort of annoying redirect. You can prevent the event from bubbling up the DOM by calling e.stopPropagation().

Our column example won't do much without the drop event in place, but before we do that, an immediate improvement is to use dragend to remove the 'over' class from each column:

...

function handleDrop(e) {

*// this / e.target is current target element.*

if (e.stopPropagation) {

e.stopPropagation(); *// stops the browser from redirecting.*

}

*// See the section on the DataTransfer object.*

return false;

}

function handleDragEnd(e) {

*// this/e.target is the source node.*

[].forEach.call(cols, function (col) {

col.classList.remove('over');

});

}

var cols = document.querySelectorAll('#columns .column');

[].forEach.call(cols, function(col) {

col.addEventListener('dragstart', handleDragStart, false);

col.addEventListener('dragenter', handleDragEnter, false)

col.addEventListener('dragover', handleDragOver, false);

col.addEventListener('dragleave', handleDragLeave, false);

col.addEventListener('drop', handleDrop, false);

col.addEventListener('dragend', handleDragEnd, false);

});

Result:

A

B

C

If you've been following closely up until now, you may notice that our example still doesn't drop the column as expected. Enter the DataTransfer object.

**The DataTransfer object**

The dataTransfer property is where all the DnD magic happens. It holds the piece of data sent in a drag action. dataTransfer is set in the dragstart event and read/handled in the drop event. Calling e.dataTransfer.setData(format, data) will set the object's content to the mimetype and data payload passed as arguments.

In our example, the data payload is set to the actual HTML of the source column:

var dragSrcEl = null;

function handleDragStart(e) {

*// Target (this) element is the source node.*

this.style.opacity = '0.4';

dragSrcEl = this;

e.dataTransfer.effectAllowed = 'move';

e.dataTransfer.setData('text/html', this.innerHTML);

}

Conveniently, dataTransfer also has a getData(format) for fetching the drag data by mimetype. Here is the modification to process the column drop:

function handleDrop(e) {

*// this/e.target is current target element.*

if (e.stopPropagation) {

e.stopPropagation(); *// Stops some browsers from redirecting.*

}

*// Don't do anything if dropping the same column we're dragging.*

if (dragSrcEl != this) {

*// Set the source column's HTML to the HTML of the column we dropped on.*

dragSrcEl.innerHTML = this.innerHTML;

this.innerHTML = e.dataTransfer.getData('text/html');

}

return false;

}

I've added a global var named dragSrcEl as a convenience to facilitate the column swap. In handleDragStart(), the innerHTML of the source column is stored in that variable and later read in handleDrop() to swap the source column and target column's HTML.

Result:

A

B

C

**Dragging properties**

The dataTransfer object exposes properties to provide visual feedback to the user during the drag process. These properties can also be used to control how each drop target responds to a particular data type.

dataTransfer.effectAllowed

Restricts what 'type of drag' the user can perform on the element. It is used in the drag-and-drop processing model to initialize the dropEffect during thedragenter and dragover events. The property can be set to the following values: none, copy, copyLink, copyMove, link, linkMove, move, all, and uninitialized.

dataTransfer.dropEffect

Controls the feedback that the user is given during the dragenter anddragover events. When the user hovers over a target element, the browser's cursor will indicate what type of operation is going to take place (e.g. a copy, a move, etc.). The effect can take on one of the following values: none, copy, link, move.

e.dataTransfer.setDragImage(imgElement, x, y)

Instead of using the browser's default 'ghost image' feedback, you can optionally set a drag icon

var dragIcon = document.createElement('img');

dragIcon.src = 'logo.png';

dragIcon.width = 100;

e.dataTransfer.setDragImage(dragIcon, -10, -10);

Result (you should see the Google logo when dragging these columns):

A

B

C

**Dragging Files**

With the DnD APIs, it is possible to drag files from the desktop to your web app in the browser window. As an extension to this idea, Google Chrome supports the ability to drag file objects out from the browser to the desktop.

**Drag-in: dragging from the desktop to the browser**

Dragging a file from the desktop is achieved by using the DnD events as other types of content. The main difference is in your drop handler. Instead of usingdataTransfer.getData() to access the files, their data will be contained in the dataTransfer.files property:

function handleDrop(e) {

e.stopPropagation(); *// Stops some browsers from redirecting.*

e.preventDefault();

var files = e.dataTransfer.files;

for (var i = 0, f; f = files[i]; i++) {

*// Read the File objects in this FileList.*

}

}

For a complete guide to dragging files from desktop to the browser, see [Using drag and drop for selecting](https://html5rocks.appspot.com/tutorials/file/dndfiles/#toc-selecting-files-dnd) in [Reading local files in JavaScript](https://html5rocks.appspot.com/tutorials/file/dndfiles/).

**Drag-out: dragging from the browser to the desktop**

For a complete guide to dragging files from the browser to the desktop, see [Drag out files like Gmail](http://www.thecssninja.com/javascript/gmail-dragout) from the CSS Ninja.

**Examples**

Here is the final product with a little more polish and a counter for each move:

A

B

C

D

The interesting thing about the column sample is that the columns are both a drag source and a drop target. A more common scenario is for the source and target elements to be different. See [html5demos.com/drag](http://html5demos.com/drag) for a demo.

**Conclusion**

No one will argue that HTML5's DnD model is complicated compared to other solutions like JQuery UI. However, any time you can take advantage of the browser's native APIs, do so! After all, that's the whole point of HTML5...which is to standardize and make available a rich set of APIs that are native to the browser. Hopefully popular libraries that implement DnD functionality will eventually include native HTML5 support by default, and fallback to a custom JS solution as needed.

**References**

* [Drag and Drop](http://www.whatwg.org/specs/web-apps/current-work/multipage/dnd.html#dnd) specification
* [Drag Operations](https://developer.mozilla.org/En/DragDrop/Drag_Operations) from MDC
* [Native Drag and Drop](http://html5doctor.com/native-drag-and-drop/) article from html5doctor
* [Drag out files like Gmail](http://www.thecssninja.com/javascript/gmail-dragout) from the CSS Ninja