# SCRIPT.ACULO.US - DRAG & DROP

The most popular feature of Web 2.0 interface is the drag and drop facility. Fortunately script.aculo.us comes with an inherent capability to support drag and drop.

To use the dragging capabilities of script.aculo.us, you'll need to load the **dragdrop** module, which also requires the **effects** module. So your minimum loading for script.aculo.us will look like this:

```
<script type="text/javascript" src="/javascript/prototype.js"></script>
<script type="text/javascript"  src="/javascript/scriptaculous.js?
load=effects,dragdrop"></script>
```

## Dragging Things Around

It is very simple to make an item draggable using script.aculo.us. It requires creating of an instance of the *Draggable* class, and identifying the element to be made draggable.

## Draggable Syntax

```
new Draggable( element, options );
```

The first parameter to the constructor identifies the element to be made draggable either as the *id* of the element, or a reference to the element. The second parameter specifies optional information on how the draggable element is to behave.

## Draggable Options

You can use one or more of the following options while creating your draggable object.

| Option | Description | Examples |
|---|---|---|
| revert | If set to *true*, the element returns to its original position when the drag ends. Also specifies whether the *reverteffect* callback will be invoked when the drag operation stops. Defaults to *false*. | Example |
| snap | Used to cause a draggable to snap to a grid or to constrain its movement. If false *default*, no snapping or constraining occurs.<br>• If it is assigned an integer x, the draggable will snap to a grid of x pixels.<br>• If an array [x, y], the horizontal dragging will snap to a grid of x pixels and the vertical will snap to y pixels.<br>• It can also be a function conforming to *Function*$x, y, draggable$ that returns an array [x, y]. | Example |
| zindex | Specifies the CSS z-index to be applied to the element during a drag operation. By default, the element's z-index is set to 1000 while dragging. | Example |
| ghosting | Boolean determining whether the draggable should be cloned for dragging, leaving the original in place until the clone is dropped. Defaults to *false*. | Example |
| constraint | A string used to limit the draggable directions, either *horizontal* or *vertical*. Defaults to *null* which means free movement. | Example |
| handle | Specifies an element to be used as the handle to start the drag | |

| | | |
|---|---|---|
| | operation. By default, an element is its own handle. | Example |
| starteffect | An effect called on element when dragging starts. By default, it changes the element's opacity to 0.2 in 0.2 seconds. | Example |
| reverteffect | An effect called on element when the drag is reverted. Defaults to a smooth slide to element's original position.Called only if *revert* is true. | Example |
| endeffect | An effect called on element when dragging ends. By default, it changes the element's opacity to 1.0 in 0.2 seconds. | Example |

## Callback Options

Additionally, you can use any of the following callback functions in the options parameter :

| Function | Description | Examples |
|---|---|---|
| onStart | Called when a drag is initiated. | Example |
| onDrag | Called repeatedly when a mouse moves, if mouse position changes from previous call. | Example |
| change | Called just as onDrag *whichisthepreferredcallback*. | Example |
| onEnd | Called when a drag is ended. | Example |

Except for the "change" callback, each of these callbacks accepts two parameters: the Draggable object, and the mouse event object.

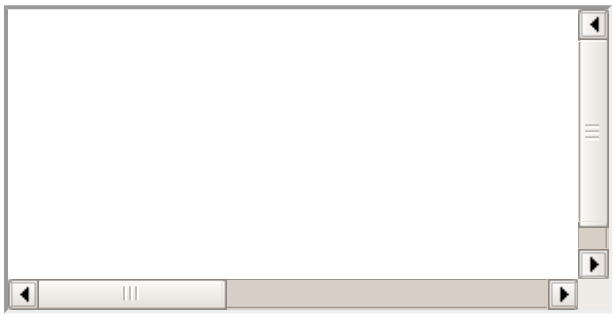
## Draggable Example

Here, we define 5 elements that are made draggable: three  elements, an <img> element, and a  element. The purpose of the three different  elements is to demonstrate that regardless of whether an element starts off with a positioning rule of static *thedefault*, relative, or absolute, the drag behavior is unaffected.

```
<html>
   <head>
      <title>Draggables Elements</title>

      <script type="text/javascript" src="/javascript/prototype.js"></script>
      <script type="text/javascript" src="/javascript/scriptaculous.js"></script>
      <script type="text/javascript">

         // Take all the elements whatever you want to make Draggable.
         var elements = ['normaldiv', 'relativediv', 'absolutediv', 'image', 'span'];

         // Make all the items drag able by creating Draggable objects
         window.onload = function() {
            elements.each(function(item) { new Draggable(item, {});});
         }

      </script>
   </head>
```

```
   <body>

      <div >
         This is a normal div and this is dragable.
      </div>

      <div >
         This is a relative div and this is dragable.
      </div>

      <div >
         This is an absolute div and this dragable.
      </div>
      <br />

      <img />

      <p>Let part <span >
         This is middle part</span> Yes, only middle part is dragable.</p>

   </body>
</html>
```

This will produce following result –



## Dropping Dragged Things

An element is converted into a drop target via a call to the *add* method within a namespace called *Droppables*.

The Droppables namespace has two important methods: *add* to create a drop target, and *remove* to remove a drop target.

## Syntax

Here is the syntax of the add method to create a drop target. The add method creates a drop target out of the element passed as its first parameter, using the options in the hash passed as the second.

```
Droppables.add( element, options );
```

The syntax for remove is even more simpler. The remove method removes the drop target behavior from the passed element.

```
Droppables.remove(element);
```

## Options

You can use one or more of the following options while creating your draggable object.

| Option | Description | Examples |
|---|---|---|
| Hoverclass | The name of a CSS class that will be added to the element while the | |

| | | |
|---|---|---|
| | droppable is active *hasanacceptabledraggablehoveringoverit*. Defaults to null. | Example |
| Accept | A string or an array of strings describing CSS classes. The droppable will only accept draggables that have one or more of these CSS classes. | Example |
| Containment | Specifies an element, or array of elements, that must be a parent of a draggable item in order for it to be accepted by the drop target. By default, no containment constraints are applied. | Example |
| Overlap | If set to 'horizontal' or 'vertical', the droppable will only react to a Draggable if its overlapping by more than 50% in the given direction. Used by Sortables, discussed in the next chapter. | |
| greedy | If true *default*, it stops hovering other droppables, under the draggable won't be searched. | Example |

## Callback Options

Additionally, you can use any of the following callback functions in the options parameter :

| Function | Description | Examples |
|---|---|---|
| onHover | Specifies a callback function that is activated when a suitable draggable item hovers over the drop target. Used by Sortables, discussed in the next chapter. | |
| onDrop | Specifies a callback function that is called when a suitable draggable element is dropped onto the drop target. | Example |

## Example

Here, the first part of this example is similar to our previous example, except that we have used Prototype's handy $A() function to convert a node list of all the <img> elements in the element with the id of draggables to an array.

```
<html>
   <head>
      <title>Drag and Drop Example</title>

      <script type="text/javascript" src="/javascript/prototype.js"></script>
      <script type="text/javascript" src="/javascript/scriptaculous.js"></script>

      <script type="text/javascript">

         window.onload = function() {

            // Make all the images draggables from draggables division.

            $A($('draggables').getElementsByTagName('img')).each(function(item) {
               new Draggable(item, {revert: true, ghosting: true});
            });

            Droppables.add('droparea', {hoverclass: 'hoverActive', onDrop: moveItem});

            // Set drop area by default  non cleared.
            $('droparea').cleared = false;
         }

         // The target drop area contains a snippet of instructional
         // text that we want to remove when the first item
```

```
         // is dropped into it.

         function moveItem( draggable,droparea){

            if (!droparea.cleared) {
               droparea.innerHTML = '';
               droparea.cleared = true;
            }

            draggable.parentNode.removeChild(draggable);
            droparea.appendChild(draggable);
         }
      </script>

      <style type="text/css">

         #draggables {
            width: 172px;
            border: 3px ridge blue;
            float: left;
            padding: 9px;
         }

         #droparea {
            float: left;
            margin-left: 16px;
            width: 172px;
            border: 3px ridge maroon;
            text-align: center;
            font-size: 24px;
            padding: 9px;
            float: left;
         }

         .hoverActive {
            background-color: #ffffcc;
         }

         #draggables img, #droparea img {
            margin: 4px;
            border:1px solid red;
         }

      </style>
   </head>
   <body>

      <div >
         <img src="/images/html.gif"/>
         <img src="/images/css.gif"/>
         <img src="/images/xhtml.gif"/>
         <img src="/images/wml_logo.gif"/>
         <img src="/images/javascript.gif"/>
      </div>

      <div >
         Drag and Drop Your Images in this area
      </div>

   </body>
</html>
```

This will produce following result –