Drag-and-Drop with jQuery: Your Essential Guide

Tutorial by Matt Doyle | Level: Intermediate | Published on 17 February 2011

Categories: *Web Development* > *JavaScript* > *jQuery*

Learn how to use jQuery, and the jQuery UI Draggable and Droppable plugins, to create drag-and-drop interfaces in your web pages. Includes a full

drag-and-drop card game example.

Dragging and dropping can be a very intuitive way for users to interact with your site or web app. People often use drag-and-drop for things like:

- Moving email messages into folders
- Reordering lists of items
- Moving objects in games around, such as cards and puzzle pieces

Drag-and-drop with JavaScript used to be very hard to do — in fact, getting a decent cross-browser version working was next to impossible. However, with modern browsers and a smattering of jQuery, drag-and-drop is now a piece of cake!

In this tutorial we'll take a look at how to create drag-and-drop interfaces with jQuery, and we'll finish with a complete drag-and-drop example: a simple number cards game for kids.

jQuery UI

To add drag-and-drop functionality to your pages, you need to include both the jQuery library and the jQuery UI plugin. jQuery UI is a fantastic plugin for jQuery that adds all sorts of useful user interface widgets, effects and behaviours — including drag-and-drop.

The easiest way to include both libraries is to use Google's CDN, as follows:

```
<head>
...

<script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jqu
<script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jqu

...
</head>
```

Making elements draggable

When you add an element to a web page — such as a `div` or an image — then that element is fixed in the page. However, using jQuery UI, it's easy to make any element draggable with the mouse.

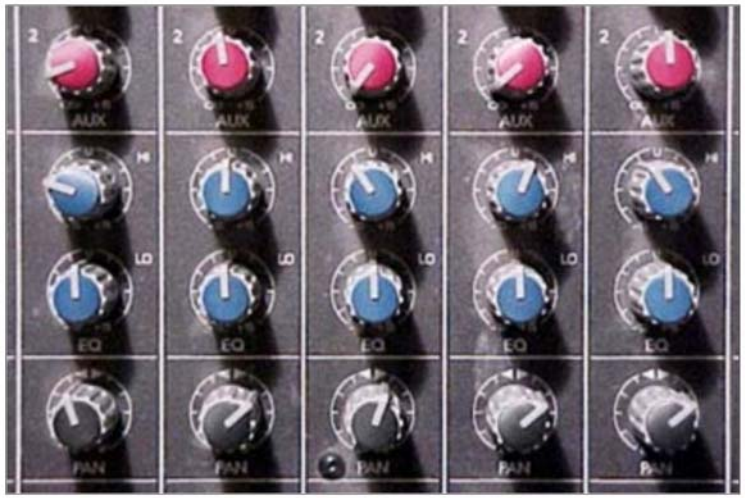
To make an element draggable, simply call the `draggable()` method on it. Here's a simple example:

```
<!doctype html>
<html lang="en">
<head>

<style>
#makeMeDraggable { width: 300px; height: 300px; background: red; }
</style>

<script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jc
<script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jc

<script type="text/javascript">

$( init );

function init() {
  $('#makeMeDraggable').draggable();
}

</script>

</head>
<body>

```

```
25  <div id="content" style="height: 400px;">
26    <div id="makeMeDraggable"> </div>
27  </div>
28
29  </body>
30  </html>
```

View Demo »

Adding draggable options



You can pass lots of options to `draggable()` to customize the behaviour of the draggable element, like this:

```
$('#makeMeDraggable').draggable( {
    option1: value1,
    option2: value2,
    ...
  } );
```

Here are some options that you'll probably want to use often:

containment

For a full list of draggable options see the jQuery UI documentation.

Let's modify our draggable square example above, and set a few options. Here's the changed code:

```
function init() {
  $('#makeMeDraggable').draggable( {
    containment: '#content',
    cursor: 'move',
    snap: '#content'
  } );
```

```
7   }
```

certain portion of the page.

You c t option to various value

View Demo »

`'parent'`	Constrains the draggable to the parent element
`'document'`	Constrains the draggable to the page
`'window'`	Constrains the draggable to the browser window
A selector	Constrains the draggable to the
Array of 4 (`[x1,y1,x`	
`cursor`	ple, you er into a
`snap`	ap the draggable element to the edges of the selected element. You can also set this option to `true` to snap the element to *any* other

Notice how the box is now constrained to, and snaps to the edges of, the `#content div`.

The cursor also changes to a move cursor while dragging.

Using a helper

Helpers are elements that are dragged instead of the original element. They are useful when

you want to leave the original element in place, but still allow the user to drag something from the element to somewhere else in the page. For example, you might want to let the user drag colours from a colour palette on top of objects to colour them.

stack If you're making a group of elements draggable — such as a set of cards — you usually want the currently-dragged element to appear on top of the other elements in the group. By setting the `stack` option to a selector that matches the group of elements, you can make sure this happens. jQuery UI adjusts the `z-index` properties of the elements so that the currently-dragged element is brought to the top.

You use the `helper` option to set a helper element for the drag operation. Possible values are:

`'original'`	The default value. The dragged element is effectively the helper, and it moves when the user drags it.
`'clone'`	Makes a copy of the element, and moves the copy instead.
A function	Lets you create a custom helper. You specify a function which accepts an `event` object and returns the markup for the helper element (or elements). This element is then moved instead of the original.

When using `'clone'` or a function to create a helper, the helper is destroyed when the drag operation stops. However, you can use the `stop` event (see Events: Responding to drags) to retrieve information about the helper — such as its position — before it's destroyed.

The following example uses a function to create a custom helper element for the drag operation. Again, this is based on the previous examples — I've just included the relevant changes here:

```
<style>
#makeMeDraggable { width: 300px; height: 300px; background: red; }
#draggableHelper { width: 300px; height: 300px; background: yellow; }
</style>

...

<script type="text/javascript">

$( init );

function init() {
  $('#makeMeDraggable').draggable( {
    cursor: 'move',
    containment: 'document',
    helper: myHelper
  } );
}

function myHelper( event ) {
  return '<div id="draggableHelper">I am a helper - drag me!</div>';
}

</script>
```

View Demo »

Events: Responding to drags

Often when the user drags an element, you want to know when the dragging has started and stopped, as well as the new position of the element. You can do this by binding event handlers to various events that are triggered by the drag operation, like this:

```
$('#makeMeDraggable').draggable( {
    eventName: eventHandler,
    ...
  } );
```

Here's a list of available events:

Your event handler function should accept 2 arguments:

- The event object (`event`).
- A jQuery UI object representing the draggable element (`ui`).

You can use the following 3 properties of the `ui` object to retrieve information about the dragged element:

`create`	Fired when the draggable element is first created by calling `draggable()`.
`start`	Fired when the user first starts dragging the element.
`drag`	Fired whenever the mouse is moved during the drag operation.
`stop`	Fired when the user lets go of the mouse button after dragging the element.

`helper`	The jQuery object representing the helper that's being dragged. If you haven't set a custom helper using the `helper` option then this object is the element itself.
`position`	An object that contains the position of the dragged element or helper, relative to the element's original position. The object has 2 properties: `left` (the x-position of the left edge of the element), and `top` (the y-position of the top edge of the element).
`offset`	An object that contains the position of the dragged element or helper, relative to the document. As with `position`, the object has `left` and `top` properties.

Let's modify our example so that it displays the final position of the dragged element, relative to the document, when the user releases the mouse button. Here's the relevant code:

```
1  <script type="text/javascript">
2
3  $( init );
4
```

```
5  function init() {
6    $('#makeMeDraggable').draggable( {
7      cursor: 'move',
8      containment: 'document',
9      stop: handleDragStop
10   } );
11 }
12
13 function handleDragStop( event, ui ) {
14   var offsetXPos = parseInt( ui.offset.left );
15   var offsetYPos = parseInt( ui.offset.top );
16   alert( "Drag stopped!\n\nOffset: (" + offsetXPos + ", " + offsetYPos + ")
17 }
18
19 </script>
```

View Demo »

Styling draggable elements

Sometimes it's nice to give elements a different look while they're being dragged. For example, you might want to highlight the dragged element, or add a drop shadow to it so it looks like it's being lifted off the page.

While an element is actually being dragged, jQuery UI gives the element a CSS class of **`ui-draggable-dragging`**. You can then add a CSS rule for this class in order to style the element while the user is dragging it.

Let's modify our simple draggable square example so that it changes from red to green while it's being dragged:

```
<style>
#makeMeDraggable { width: 300px; height: 300px; background: red; }
#makeMeDraggable.ui-draggable-dragging { background: green; }
</style>
```

View Demo »

Droppables: Accepting draggable elements

So far we've learned how to make an element draggable, so that the user can drag it around the page. We've also looked at using events to respond to drags and drops.

However, there's an easier way to deal with drops, and that is to create droppable elements. A ***droppable element*** is an element that can accept a draggable element that has been dragged and dropped onto it.

When a draggable is dropped on a droppable, the droppable fires a `drop` event. By writing an event handler function for the `drop` event, you can determine which draggable was dropped onto the droppable.

You can also use the `over` and `out` events to determine when a draggable is being dragged over or out of a droppable. For more on droppable events, see the jQuery UI

docs.

To make an element droppable, you call — you guessed it — `droppable()`.

Let's modify our draggable square example to include a droppable element that the square can be dropped onto:

```
<!doctype html>
<html lang="en">
<head>

<style>
#makeMeDraggable { float: left; width: 300px; height: 300px; background: re
#makeMeDroppable { float: right; width: 300px; height: 300px; border: 1px s
</style>

<script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jq
<script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jq

<script type="text/javascript">

$( init );

function init() {
  $('#makeMeDraggable').draggable();
  $('#makeMeDroppable').droppable( {
    drop: handleDropEvent
  } );
}

function handleDropEvent( event, ui ) {
```

```
25    var draggable = ui.draggable;
26    alert( 'The square with ID "' + draggable.attr('id') + '" was dropped ont
27  }
28
29  </script>
30
31  </head>
32  <body>
33
34  <div id="content" style="height: 400px;">
35
36    <div id="makeMeDraggable"> </div>
37    <div id="makeMeDroppable"> </div>
38
39  </div>
40
41  </body>
42  </html>
```

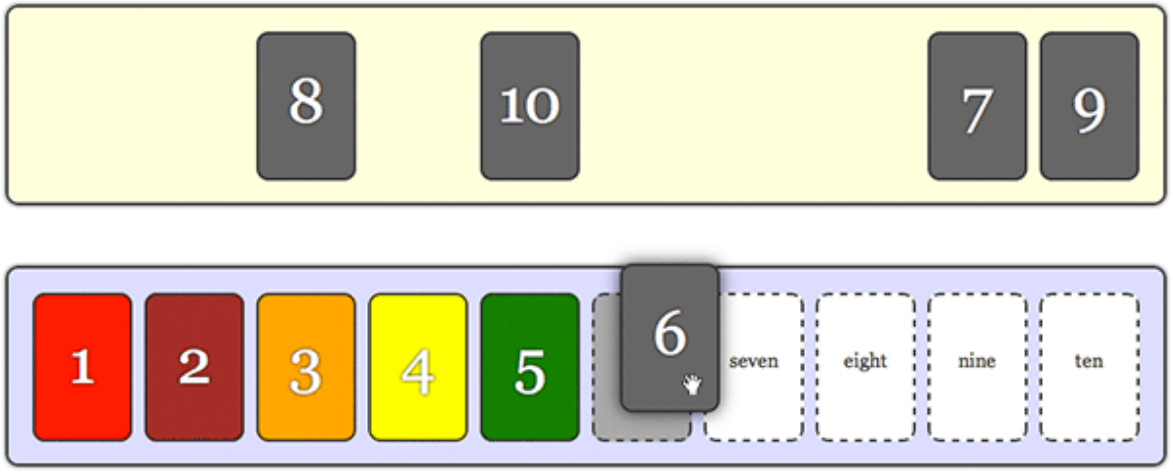
View Demo »

As you can see, we've created an event handler function called `handleDropEvent()` to respond to the droppable's `drop` event. As with draggable events, the handler needs to accept an `event` and a `ui` object.

The `ui` object has a `draggable` property, which is the draggable element that was dragged onto the droppable. Our function uses this object, along with the jQuery `attr()` method, to

retrieve the ID of the dropped element (`"makeMeDraggable"`) and display it using `alert()`.

Complete example: A drag-and-drop number cards game



A lot of the above concepts are easier to grasp when you see them working in a real example. So let's make one!

We're going to build a simple number cards game for young kids. The user is presented with 10 number cards in random order, and 10 slots to drop the cards onto. The object of the game is to drop all 10 cards onto the correct slots. Try out the finished game by clicking the button below:

View Demo »

When playing the game, you can view the page source to see all the markup and JavaScript code.

Step 1: The markup

The HTML for our game is straightforward:

```
<!doctype html>
<html lang="en">
<head>

<title>A jQuery Drag-and-Drop Number Cards Game</title>
<meta http-equiv="Content-Type" content="text/html;charset=utf-8">
<link rel="stylesheet" type="text/css" href="style.css">

<script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/j
<script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/j

<script type="text/javascript">

// JavaScript will go here
```

```
15
16  </script>
17
18  </head>
19  <body>
20
21  <div id="content">
22
23    <div id="cardPile"> </div>
24    <div id="cardSlots"> </div>
25
26    <div id="successMessage">
27      <h2>You did it!</h2>
28      <button onclick="init()">Play Again</button>
29    </div>
30
31  </div>
32
33  </body>
34  </html>
```

In the `head` area we've included the jQuery and jQuery UI libraries, and linked to a `style.css` file which we'll build in Step 4. We've also added a `script` element for our JavaScript code (which we'll write in a moment). In the `body` we have:

- A `"#content"` `div` to contain the game
- A `"#cardPile"` `div` that will contain the set of 10 unsorted cards
- A `"#cardSlots"` `div` that will contain the 10 slots to drop the cards into
- A `"#successMessage"` `div` to display a "You did it!" message, along with a button to

play again. (We'll hide this initially using JavaScript when the page loads.)

Step 2: The `init()` function

Our first chunk of JavaScript code sets up the game:

```
var correctCards = 0;
$( init );

function init() {

  // Hide the success message
  $('#successMessage').hide();
  $('#successMessage').css( {
    left: '580px',
    top: '250px',
    width: 0,
    height: 0
  } );

  // Reset the game
  correctCards = 0;
  $('#cardPile').html( '' );
  $('#cardSlots').html( '' );

  // Create the pile of shuffled cards
  var numbers = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ];
  numbers.sort( function() { return Math.random() - .5 } );

  for ( var i=0; i<10; i++ ) {
    $('<div>' + numbers[i] + '</div>').data( 'number', numbers[i] ).attr( '
```

```
26        containment: '#content',
27        stack: '#cardPile div',
28        cursor: 'move',
29        revert: true
30      } );
31    }
32
33    // Create the card slots
34    var words = [ 'one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eig
35    for ( var i=1; i<=10; i++ ) {
36      $('<div>' + words[i-1] + '</div>').data( 'number', i ).appendTo( '#card
37        accept: '#cardPile div',
38        hoverClass: 'hovered',
39        drop: handleCardDrop
40      } );
41    }
42
43  }
```

Let's break the above code down:

1. **Initialize the `correctCards` variable**

   We first set up a variable called `correctCards` to track the number of correctly-dropped cards. When this reaches 10 then the user has won the game.

2. **Call `init()` when the document is ready**

   We use `$( init )` to call our `init()` function once the DOM has loaded.

3. **Hide the success message**

   Within the `init()` function itself, we first hide the `#successMessage div`, and reduce its width and height to zero so that we can make it "pop out" when the game is won.

4. **Reset the game**

   Since `init()` can be called after a game has already been played, we set `correctCards` to zero so that the game can begin again, and also clear out the `#cardPile` and `#cardSlots div`s so that we can populate them afresh.

5. **Create the pile of shuffled cards**

   To create the pile of cards, we first make an array, `numbers`, containing the numbers 1 to 10, then sort the array with a randomizing sort function to shuffle the numbers.

   Then we loop through each element in the `numbers` array, creating a `div` card element for each number. Inside the `div` we place the number, so that it appears on the card in the page. Once we've created the `div` object, we store the card's number in a `'number'` key inside the object by using jQuery's `data()` method.

   We also give the card `div` an `id` of `'card`*n*`'`, where *n* is the card number. This lets us give each card a different colour in the CSS. We then append the card to the `#cardPile div`.

   The interesting bit in the loop is, of course, the call to the `draggable()` method. This:

   - Makes the card draggable
   - Uses `containment` to constrain it to the `#content div`
   - Uses `stack` to ensure that the dragged card always sits on top of the other cards
   - Uses `cursor` to turn the mouse cursor into a move cursor during the drag, and
   - Sets the `revert` option to `true`. This makes the card slide back to its initial position when dropped, so that the user can try again. We'll turn this option off when the user has dragged the card to the correct slot, as we'll see in a moment.

6. **Create the card slots**

The last part of the `init()` function creates the 10 slots to drag the cards onto. We create an array called `words` that contains the words "one" to "ten", then loop through the elements of the `words` array, creating a slot `div` for each number. As before, we use `data()` to record the number of the slot, so we can test if the user has dragged the correct card to the correct slot, and we append the slot to the `#cardSlots div`.

This time, we call the `droppable()` method so that the slot can receive a draggable card. We use the `accept` option with the selector `"#cardPile div"` to ensure that the slot will only accept our number cards, and not any other draggable element. The `hoverClass` option adds a CSS class to a droppable when a draggable is hovered over it — we use this option to add a class of `'hovered'` to the element, which we'll highlight using CSS. Finally, we set up a `drop` event handler called `handleCardDrop()`, which is triggered when the user drops a card on the droppable. We'll write this handler next.

You can also use a function with `accept` in order to limit the types of draggable that your droppable accepts. For more details on the options available with droppable, check out the docs.

Step 3: The `handleCardDrop()` event handler function

The last piece of JavaScript we'll write is the event handler for our droppables' `drop` events, `handleCardDrop()`:

```
1  function handleCardDrop( event, ui ) {
2    var slotNumber = $(this).data( 'number' );
3    var cardNumber = ui.draggable.data( 'number' );
```

```
4
5     // If the card was dropped to the correct slot,
6     // change the card colour, position it directly
7     // on top of the slot, and prevent it being dragged
8     // again
9
10    if ( slotNumber == cardNumber ) {
11      ui.draggable.addClass( 'correct' );
12      ui.draggable.draggable( 'disable' );
13      $(this).droppable( 'disable' );
14      ui.draggable.position( { of: $(this), my: 'left top', at: 'left top' }
15      ui.draggable.draggable( 'option', 'revert', false );
16      correctCards++;
17    }
18
19    // If all the cards have been placed correctly then display a message
20    // and reset the cards for another go
21
22    if ( correctCards == 10 ) {
23      $('#successMessage').show();
24      $('#successMessage').animate( {
25        left: '380px',
26        top: '200px',
27        width: '400px',
28        height: '100px',
29        opacity: 1
30      } );
31    }
32
```

Let's work through this function:

1. **Grab the slot number and card number**

   The first thing we do is grab the number of the slot that the card was dropped on, as well as the number of the dropped card, so that we can see if they match.

   As our function is an event handler for the droppable element, we can retrieve the element via the `this` variable. We then wrap the element inside the `$()` function to turn it into a jQuery object, then use the `data()` method to retrieve the value of the `'number'` key, and store the value in `slotNumber`.

   Similarly, the `draggable` property of the `ui` object contains the jQuery object representing the dragged card. By reading its `'number'` key using `data()`, we get the card number, which we store in `cardNumber`.

   > Remember that we stored the slot and card numbers using `data()` in the `init()` function in Step 2.

---

2. **If the card and slot match, drop the card onto the slot**

   If the 2 numbers match then the correct card was dropped onto the slot. First we add a `'correct'` CSS class to the card so that we can change its colour via CSS. We then disable both the draggable and droppable by calling their `disable` methods. This prevents the user from being able to drag this card, or drag another card onto this slot, again.

   > Find out more about draggable/droppable methods in the draggable and droppable docs.

After disabling the card and slot, we position the card directly on top of the slot by calling the `position()` method. This handy method lets you position any element relative to practically anything else, and also plays nicely with draggable and droppable elements. In this case, we're positioning the card (`ui.draggable`)'s top left corner (`my: 'left top'`) on top of the slot (`$this`)'s top left corner (`at: 'left top'`).

Finally we set the card's `revert` option to `false`. This prevents the card from being pulled back to its initial position once it has been dropped. We also increment the `correctCards` variable to keep track of the number of correctly-dropped cards.

3. **Check for a win**

   If `correctCards` equals 10, we have a winner! We show the success message, then animate it from zero width and height up to its full width and height, creating a zooming effect. The success message includes a Play Again button in the markup that triggers the `init()` function when clicked, thereby starting a new game.

Step 4: The CSS

The last stage of creating our card game is to style the page, cards and slots using CSS. Here's the complete style.css file:

```
/* Add some margin to the page and set a default font and colour */

body {
  margin: 30px;
  font-family: "Georgia", serif;
  line-height: 1.8em;
  color: #333;
}

```

```
/* Give headings their own font */

h1, h2, h3, h4 {
  font-family: "Lucida Sans Unicode", "Lucida Grande", sans-serif;
}

/* Main content area */

#content {
  margin: 80px 70px;
  text-align: center;
  -moz-user-select: none;
  -webkit-user-select: none;
  user-select: none;
}

/* Header/footer boxes */

.wideBox {
  clear: both;
  text-align: center;
  margin: 70px;
  padding: 10px;
  background: #ebedf2;
  border: 1px solid #333;
}

.wideBox h1 {
  font-weight: bold;
  margin: 20px;
  color: #666;
  font-size: 1.5em;
```

```
}

/* Slots for final card positions */

#cardSlots {
  margin: 50px auto 0 auto;
  background: #ddf;
}

/* The initial pile of unsorted cards */

#cardPile {
  margin: 0 auto;
  background: #ffd;
}

#cardSlots, #cardPile {
  width: 910px;
  height: 120px;
  padding: 20px;
  border: 2px solid #333;
  -moz-border-radius: 10px;
  -webkit-border-radius: 10px;
  border-radius: 10px;
  -moz-box-shadow: 0 0 .3em rgba(0, 0, 0, .8);
  -webkit-box-shadow: 0 0 .3em rgba(0, 0, 0, .8);
  box-shadow: 0 0 .3em rgba(0, 0, 0, .8);
}

/* Individual cards and slots */

#cardSlots div, #cardPile div {
```

```
  float: left;
  width: 58px;
  height: 78px;
  padding: 10px;
  padding-top: 40px;
  padding-bottom: 0;
  border: 2px solid #333;
  -moz-border-radius: 10px;
  -webkit-border-radius: 10px;
  border-radius: 10px;
  margin: 0 0 0 10px;
  background: #fff;
}

#cardSlots div:first-child, #cardPile div:first-child {
  margin-left: 0;
}

#cardSlots div.hovered {
  background: #aaa;
}

#cardSlots div {
  border-style: dashed;
}

#cardPile div {
  background: #666;
  color: #fff;
  font-size: 50px;
  text-shadow: 0 0 3px #000;
}
```

```
106
107 #cardPile div.ui-draggable-dragging {
108   -moz-box-shadow: 0 0 .5em rgba(0, 0, 0, .8);
109   -webkit-box-shadow: 0 0 .5em rgba(0, 0, 0, .8);
110   box-shadow: 0 0 .5em rgba(0, 0, 0, .8);
111 }
112
113 /* Individually coloured cards */
114
115 #card1.correct { background: red; }
116 #card2.correct { background: brown; }
117 #card3.correct { background: orange; }
118 #card4.correct { background: yellow; }
119 #card5.correct { background: green; }
120 #card6.correct { background: cyan; }
121 #card7.correct { background: blue; }
122 #card8.correct { background: indigo; }
123 #card9.correct { background: purple; }
124 #card10.correct { background: violet; }
125
126
127 /* "You did it!" message */
128 #successMessage {
129   position: absolute;
130   left: 580px;
131   top: 250px;
132   width: 0;
133   height: 0;
134   z-index: 100;
135   background: #dfd;
136   border: 2px solid #333;
137   -moz-border-radius: 10px;
```

```
    -webkit-border-radius: 10px;
    border-radius: 10px;
    -moz-box-shadow: .3em .3em .5em rgba(0, 0, 0, .8);
    -webkit-box-shadow: .3em .3em .5em rgba(0, 0, 0, .8);
    box-shadow: .3em .3em .5em rgba(0, 0, 0, .8);
    padding: 20px;
  }
```

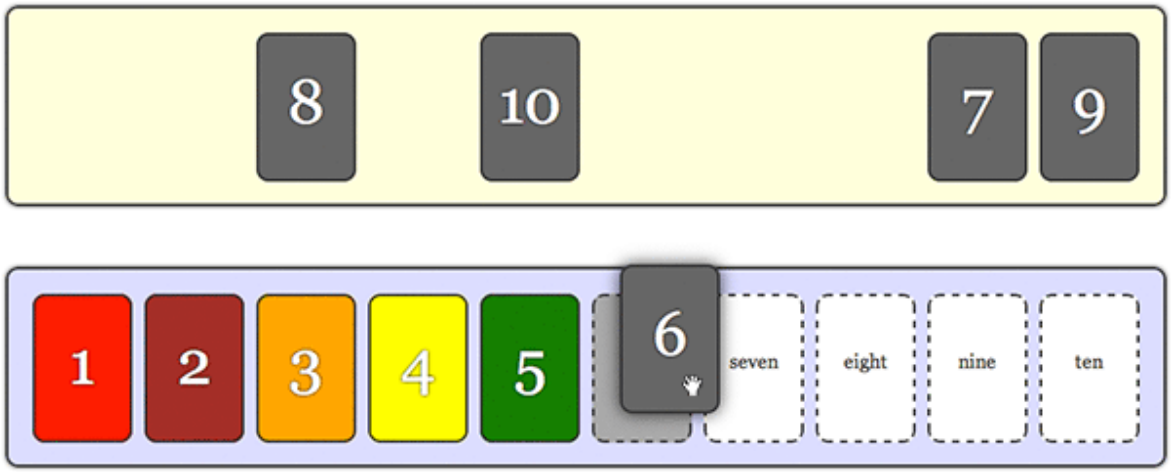
I won't go into detail with this file, since most of it is either self-explanatory or not relevant to the tutorial. Some rules of interest though:

- **`#cardSlots and #cardPile`** — the rectangles containing the slots and unsorted cards respectively — are given a 2-pixel dark grey border with rounded corners and a drop shadow.
- **`#cardSlots div and #cardPile div`** select the slots and cards themselves. They're floated left so they all line up in a horizontal row, and given some padding and the same style border as the `#cardSlots` and `#cardPile` rectangles. Additionally, the card slots are given a dashed border, and the cards are given a nice big font for the numbers. The `text-shadow` property is used to put an outline around the numbers.
- **`#cardSlots div.hovered`** is triggered when a card hovers over a slot, thanks to the `hoverclass: 'hovered'` option we added in Step 2 above. We give this selector a darker grey background, so that the slot is highlighted when a card is hovered over it. This helps the user identify which slot they're dropping the card onto.
- **`#cardPile div.ui-draggable-dragging`** is triggered when a card is being dragged (see Styling draggable elements). We give this selector a drop shadow so the card appears to be floating above the page.
- **`#card1.correct through #card10.correct`** select the individual cards once they've been correctly placed, due to the call to `addClass()` that we made inside

`handleCardDrop()` (see Step 3 above). We give each correct card a different colour to create a nice rainbow effect. We can do this because we gave each card a unique `id` in Step 2.

- Finally, **`#successMessage`** styles the "You did it!" box, positioning it centrally in the page and giving it a green background, curvy border and box shadow. We set its width and height to zero initially, so we can zoom it in when the user wins the game.

All done!



We've now built our drag-and drop card game. Try it out again if you like:

View Demo »

In this tutorial you learned how to use the jQuery and jQuery UI libraries to add drag-and-drop functionality to your web pages. You looked at:

- The Draggable plugin, which makes it easy to turn any element into a draggable widget in the page
- How to specify options for draggable elements
- Using a helper element, which is dragged instead of the draggable element
- Working with draggable events to get information about dragged elements
- Styling elements while they're being dragged, and
- The Droppable plugin, which lets you easily identify and handle dropped elements.

Finally, we brought all this knowledge together to build a simple drag-and-drop card game for kids.

I hope you found this article helpful. There's a lot more to draggables and droppables — and jQuery UI, for that matter — than we've covered here. For more information, see the jQuery UI website.

Happy coding!

Link HTML: <a href="http://www.elated.c

Your Email Address... Send

URL only: http://www.elated.com/articl

Related articles

- jQuery Mobile 1.1: Smoother, Faster, Nicer
- Ajax with jQuery: A Beginner's Guide
- jQuery Mobile Masterclass: Build a Simple, Attractive Twitter App for iPhone
- Cover Flow Remade with CSS and jQuery
- How to Make a Slick Ajax Contact Form with jQuery and PHP
- How to Make an Elegant Sliding Image Gallery with jQuery
- jQuery Mobile: What Can It Do for You?
- Create Smooth Rotatable Images with CSS3 and jQuery
- A Snazzy Animated Pie Chart with HTML5 and jQuery

Responses to this article

20 most recent responses *(oldest first)*:

Chris Ward 03-Apr-12 09:13

@survivant

I've felt that pain.

This took it away for me within minutes of discovering it :

http://furf.com/exp/touch-punch/draggable.html

Hope it works for you too.

Regard,
Chris

k_ace 10-Apr-12 14:40

I would like to ask if there is a way to use the objects as a part of form submission. For example, the user will only know the correct placement of the draggable object after clicking the SUBMIT button instead of real time results.

RSugelSr 10-Apr-12 15:38

k_ace,

I used Elated's D & D as the basis for an eLearning activity. Not a form submission but the same concept. Had to write additional methods, but the basic premise is to compare a drag's final position to the correct position. After clicking the "Submit" button, it validates visually indicating which items were placed correctly/incorrectly and then displays a "Solution" button which moves all drags to their correct positions.

kyleross 13-Apr-12 09:06

Hi,

Firstly I love the plugin, it works great for what I want it to do!

I have used the card-draggable code as I have created a system to approve designs based on the user dragging Object A to Object B then firing a SQL statement.

My problem is when adding the divs, namely:

```
<div id="cardPile"> </div>
<div id="cardSlots"> </div>
<div id="successMessage"></div>
```

I can only see 1 of the drag container, but my project pulls the main container - *with the 'cardpile' inside* based on the amount of entries in the database. i.e there are 8 designs, it pulls 8 containers, but this plugin only pulls into the first one. I assume it's because of the ID?

Can you please help with how to have multiple entities of these divs on one page?

Appreciate any assistance!

minttoothpick 18-Apr-12 13:47

@ChrisWard, your information about the Model 'class' for handling a grid-based game

layout is something I've been trying to figure out myself for the past week or so... I just didn't realize what exactly I was looking for!

That was a great example, but I was wondering if you know of any other resources that go into the concept a bit deeper; I haven't come across any tutorials or training that really lays it out.

@matt, this article has been invaluable as well!

Chris Ward 19-Apr-12 06:18

@minttoothpick - thank you very much, I am pleased if it helped in any way.

I don't really know of any other resources. I didn't find anything on the web to get me going down this route.

There were three influences that drove me to use a "model" -

a) I am using Grails and it's clear distinction between Model, View and Controller were at the front of my mind at the time. I've done a lot of GUI application work over the years to tend to use this approach as default

b) Complexity of what I wanted to store - my application needs to maintain pretty darn complex state and I needed to have it in an actual data structure rather than rely on the DOM states.

c) Load/Save - My application requires me to be able to persist my visual representation and be able to reload it. Once you have your stuff represented in a data structure (the "model" in JavaScript with the member fields and the getters and setters and convenience functions) it's not too difficult to puke that out in something like JSON. Once you've done it in that direction (saving) you can do the other direction (loading). I tend to have a function

called

```
inflate(jsonString)
```

in my models and it just takes the JSON string input and calls a library function to reestablish the proper model instance from it.

I don't know if this helps at all. My general advice would be to think about a model/data structure that would hold all the state(s) your thing needs. Maybe even write some tests to use the model until you're happy with it. At that point you can move to doing the visual stuff on top.

One of the first things the appealed to me with jQuery was the ability to attach data to visual objects using

```
$(locator).data(your-bit-of-data)
```

Once you do that it makes it nice and clean to access this data inside a drop method (if you're using drag and drop) and use that as the param for your calls to your model functions.

Not sure if this makes it clearer or most confusing!

Chris

minttoothpick 19-Apr-12 07:40

@ChrisWard - Thanks again for your comments. I have only been working with JavaScript for several months, and it was in the past two weeks that I started using jQuery. Discovering the .data() method has been very helpful.

Currently I am putting together a checkers board game. I started by implementing the <canvas> element and tried a couple related JS libraries for working with it, but it proved too complicated. Now I'm working with jQuery and elements to create the game board, but I am still figuring out how to best reference the individual cells for the game logic.

I haven't delved into JSON yet but I've been hearing about it a lot.

Anyways, I appreciate that you have taken the time to provide such valuable information!

nicolasmoise 19-Apr-12 15:53

Hi, your guide is wonderful and very helpful.

However, I am having a problem. I am using a function to create a helper (instead of clone), but I am trying to pass a variable to this helper. More specifically, I am trying to pass the inner html of the original draggable element to the helper element.

Here is my very lame attempt:

```
function catHelper(event, ui){
    var name=ui.draggable.html();
```

```
    return '<div class="cat_helper">'+name+'</div>';
}
```

Chris Ward 20-Apr-12 04:10

@minttoothpick

One way to do what you need is to have s for the black and white pieces and 8x8 s for the playing board.

If you want to identify the colour of a piece (or counter or whatever you want to call it) you could apply a .data("black") or .data("white"). However - you might want to use the css class attribute since you'll probably be needed to style these s to have a black or white image.

Then on the board each of the squares could have data in the range .data("1,1") ... .data("8,8").

If you make the board squares droppable, and the playing piece draggable then when you drop a piece onto a square you should be able to work out the piece colour and the grid reference in the handleDrop function (which gives you a reference to both the draggable and the droppable)

Once you have these you could call your model method like

model.setCell(x, y, pieceColour)

Something along those lines should work I'd have thought.

matt 27-Apr-12 19:46

@minttoothpick: If you want to learn JSON then you might enjoy my tutorial:

http://www.elated.com/articles/json-basics/

@nicolasmoise: Off the top of my head, you probably want to create a closure, passing in the draggable element so that your function can access it.

---

xkater 18-May-12 02:11

---

thanks for the great dnd tutorial, Matt!
In the tutorial, the numbers get located randomly in the Card Pile, and the Card Slots are located 1-10, left to right --- Card "1" will always drop on the first Slot.
I want to build a game where you don't know which 'Correct' slot the number should drop into - in other words, so slot "1" (array position 0) locates randomly in the row of 10 Card Slot locations, instead of always in the first (left-most) location, but i've had no luck 'sorting' the Slots to any other position.

any ideas?

tia

---

danahartweg 02-Jun-12 17:05

First of all, this is a great introduction to jQuery UI and drag and drop! I found it immensely helpful when getting up to speed and implementing it in to my project.

One issue I came across was reassigning the parent div of the draggable object when a drop occurred. You can get some positioning weirdness.

If anyone runs across the same problem, you can see my solution here:

http://eightbitswitch.com/2012/06/jquery-ui-drag-and-drop-to-change-parent-div/

---

matt 15-Jun-12 02:20

@xkater: Can you not just call the sort() method on the words array, in the same way as the numbers array?

@danahartweg: Thanks for posting that solution

---

crush123 03-Jul-12 10:47

Excellent article, very helpful.

I have a project in mind, so will try to get to grips with this

Taking the card example as a starting point, I would like to add multiple instances of the same card(s) and stack them on top of each other, (or nearly on top, so you can tell its a stack).

Grateful for an approach

---

mnaputi 14-Aug-12 21:26

Great article.

I would like to set this up for my students where they can practice vocabulary, but I am still relatively new to jquery.

How would I go about replacing the numerals with images so that an image could be dragged onto the correct word?

Thanks in advance!

---

sameera207 09-Oct-12 22:40

Superb article, everything is crystal clear, thanks a lot for finding time to compose this

cheers

sameera

Foreverns 18-Oct-12 04:06

That's cool

---

piapoco 09-Nov-12 13:00

Hi, Matt. I've found your code great, but I've run into a problem with Chrome: when the user zooms in or out ,the cards refuse to drop in their intended positions. Have you developed a solution yet? Thanks a lot for the code and for your answer!

---

matt 21-Nov-12 21:29

@piapoco: Looks like a general problem with any jQuery UI draggable. Try zooming in then dragging "I only snap to the big box" - it no longer snaps to the big box edges: http://jqueryui.com/draggable/#snap-to

---

goldilaks 26-Nov-12 13:48

Thanks so much for the guide. This was my first try at using drag and drop (and pretty much jQuery in general) and you walked me through it easily. Great article and helpful links to the documentation.

View all 66 responses »

Post a response

Want to add a comment, or ask a question about this article? Post a response.

To post responses you need to be a member. Not a member yet? Signing up is free, easy and only takes a minute. Sign up now.

Top of Page