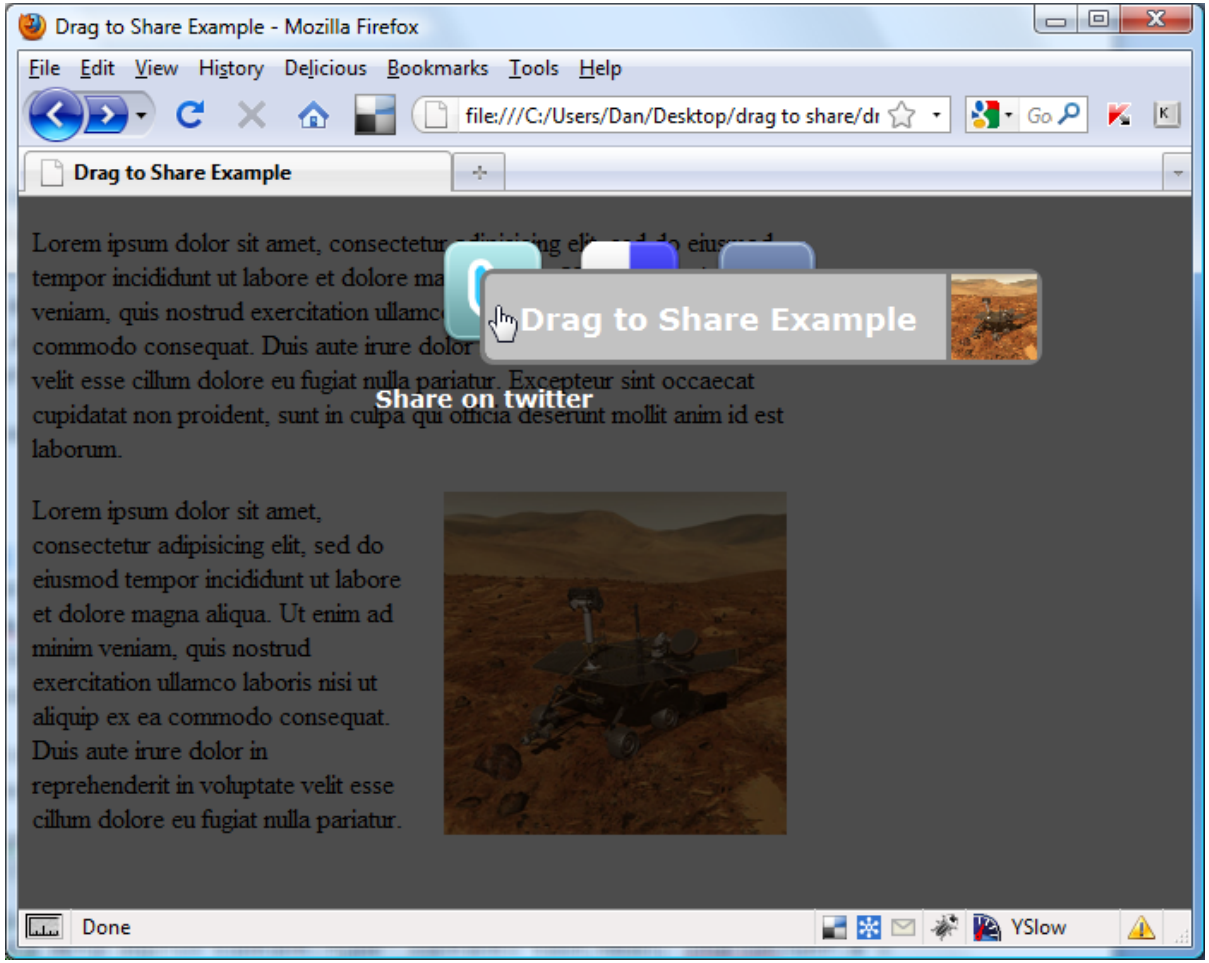
Drag to Share

Dan Wellman on Oct 19th 2009 with 109 Comments and 0 Reactions

We've all seen the brilliant functionality on Mashable where news stories and interesting articles can be shared to social networking sites; the functionality is driven by the images accompanying the articles; you click and hold on an image and can then drag it into a toolbar to share it. It's brilliant and intuitive, and in this article I'm going to show you how we can replicate this behavior with jQuery and jQuery UI.

The following screenshot shows what we'll have at the end of the tutorial:



Getting Started

The latest version of jQuery comes with jQuery UI and in this example we only need the core, draggable and droppable components, so make sure only these are selected in the download builder. Once the jQuery UI archive has been downloaded, unpack the js folder from the archive (we don't need the development bundle or CSS framework in this example) in a working folder.

Now let's create a basic page, with some text and an image on it, to showcase the behaviour; create the following new page in your code editor:

view plaincopy to clipboardprint?

```
1. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
```

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>Drag to Share Example</title>
    <link rel="stylesheet" type="text/css" href="dragToShare.css">
  </head>
  <body>
    <div id="content">
      <p>Lorem ipsum dolor...</p>
      <img src="rover.png" alt="Mars Rover">
      <p>Lorem ipsum dolor...</p>
    </div>
    <script type="text/javascript" src="js/jquery-1.3.2.min.js"></script>
    <script type="text/javascript" src="js/jquery-ui-1.7.2.custom.min.js"></script>
  </body>
</html>
```

Save this as dragToShare.html in the folder with the js folder in it. Here we've added our layout/example text and an image, both within a container. We're linking to the jQuery and jQuery UI source files at the bottom of the <body> and a custom style sheet in the <head>. We don't need many styles at this point as there isn't much on the page to actually style, but let's add it next anyway with some basic styles for the page elements in it; in a new file in your code editor add the following:

view plaincopy to clipboardprint?

```
#content { width:440px; }
#content img { float:rightright; margin-left:20px; }
```

Save this tiny file as dragToShare.css in the same folder as our HTML page. Don't worry, we'll be adding some more styles to the style sheet very shortly. Our example page should look like this at this point:

Making the Image Draggable

We need to make the image draggable, which we can do with jQuery UI, add the following <script> element after the others:

view plaincopy to clipboardprint?

```
1. <script type="text/javascript">
2.   $(function() {
3.     var images = $("#content img"),
```

```
4.       title = $("title").text() || document.title;
5.     //make images draggable
6.     images.draggable();
7.   });
8. </script>
```

That's all we need to do! We just cache the selector for the element(s) that we'd like to make draggable, and call the draggable() method on the resulting collection of elements. Now by clicking and holding the mouse button, the image in our example can be dragged around the page. The image will be made draggable as soon as the document is loaded as our code is wrapped in the $(function(){}) shortcut.

As well as caching the selector that returns our images, we're also caching a selector that stores the title of the page. IE returns an empty string when using jQuery to retrieve the page title so we revert to document.title in this case.

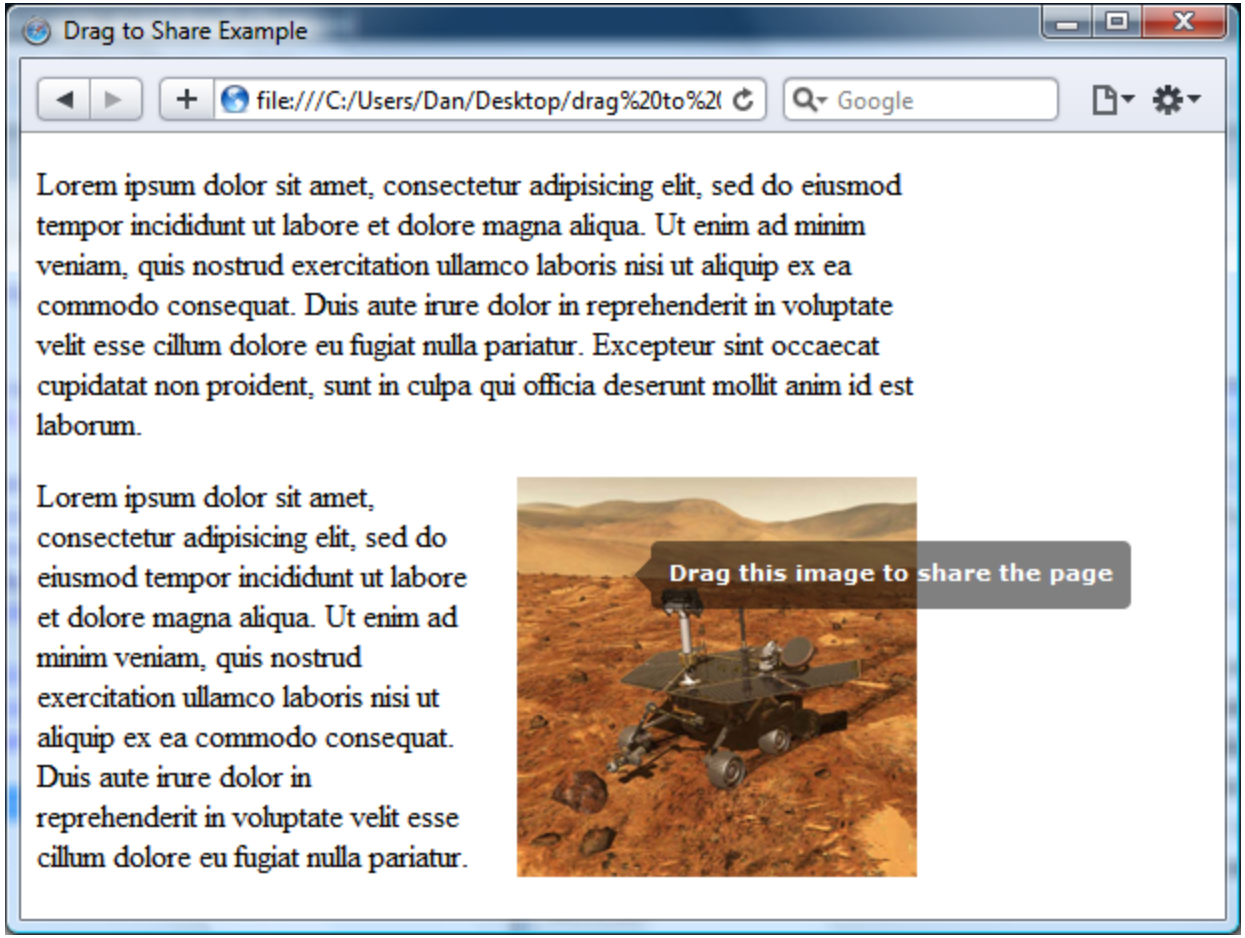
We still have a lot to do before we're done though; what we need to do first is to let the visitor know that dragging the image does something. Firstly we can use a little CSS to set the 'move' mouse pointer when we hover over the image; add the following line to the end of dragToShare.css:

view plaincopy to clipboardprint?

```
1. .ui-draggable { cursor:move; }
```

We're targeting the class .ui-draggable which is added by jQuery UI with this style so that the image will only inherit the move cursor if the image is made draggable, which won't happen if JavaScript is switched off or otherwise unavailable. Using the class name instead of the :hover pseudo class is also much better for cross-browser compatibility.

Information Overlay



To really make it obvious that dragging the image does something, we can also add a tooltip to explicitly tell the visitor what to do; after the draggable() method add the following new code:

view plaincopy to clipboardprint?

```
1. var createTip = function(e) {
2.   //create tool tip if it doesn't exist
3.   ($("#tip").length === 0) ? $("<div>").html("<span>Drag this image to share the page<\/span><span class='arrow'>
     <\/span>").attr("id", "tip").css({ left:e.pageX + 30, top:e.pageY - 16 }).appendTo("body").fadeIn(2000) : null;
4. };
5. images.bind("mouseenter", createTip);
6. images.mousemove(function(e) {
```

```
    //move tooltip
    $("#tip").css({ left:e.pageX + 30, top:e.pageY - 16 });
  });
  images.mouseleave(function() {
    //remove tooltip
    $("#tip").remove();
  });
```

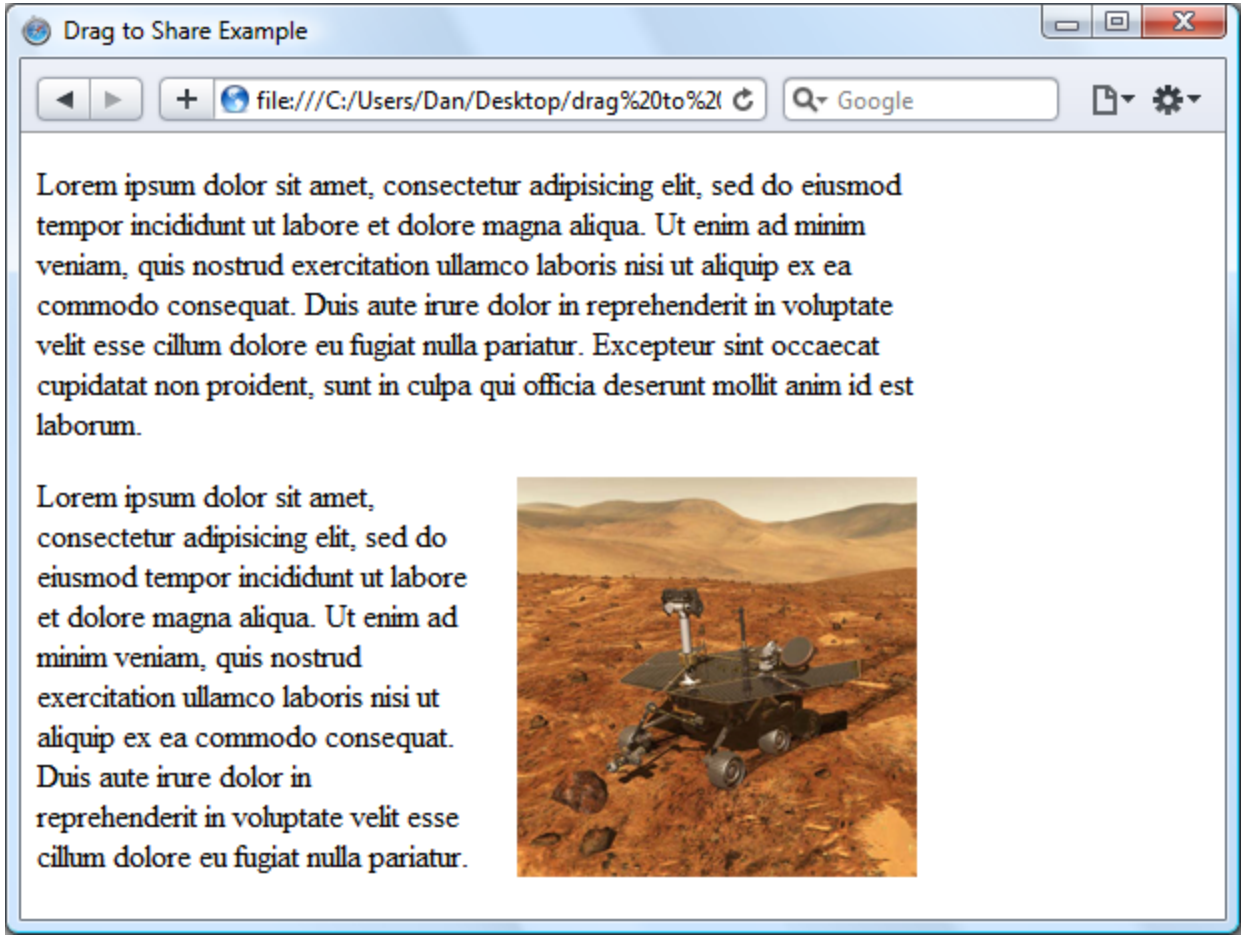
We've basically added three new event handlers to our code; the first event handler is the createTip function, which is defined as a variable and passed to jQuery's bind() method along with the string mouseenter specifying the event. The next two functions are anonymous and are passed inline to jQuery's mousemove() and mouseleave() helper methods.

In the createTip function we first check whether the tooltip already exists by seeing if a selector for it has a length of 0. If it does (have a length of 0) we know it doesn't exist and can then create it. We set the innerHTML of the tooltip so that it features a span containing the message to the visitor, and a second empty span which we'll use for a little decoration when we add the additional CSS in a moment.

We give the tooltip an id so that we can select it efficiently later on and set its CSS left and top properties using the pageX and pageY properties from the event object (e) which is passed to our function automatically. We then append the tooltip to the body of the page and fade it in slowly. To avoid HTML errors, we need to escape the forward-slashes in the raw HTML we add to the tooltip.

The mousemove function is used to make the tooltip track with the pointer, so when the pointer moves, the tooltip moves with it. We use the css method again as well as the pageX and pageY properties. Finally, the mouseleave function simply removes the tooltip from the page.

Styling the Tooltip



Now we can add some CSS for our tooltip; in the interests of progressive enhancement we'll use the sexy CSS3 rgba style to make out tooltip semi-transparent in capable browsers; at the bottom of dragToShare.css add the following new selectors and rules:

view plaincopy to clipboardprint?

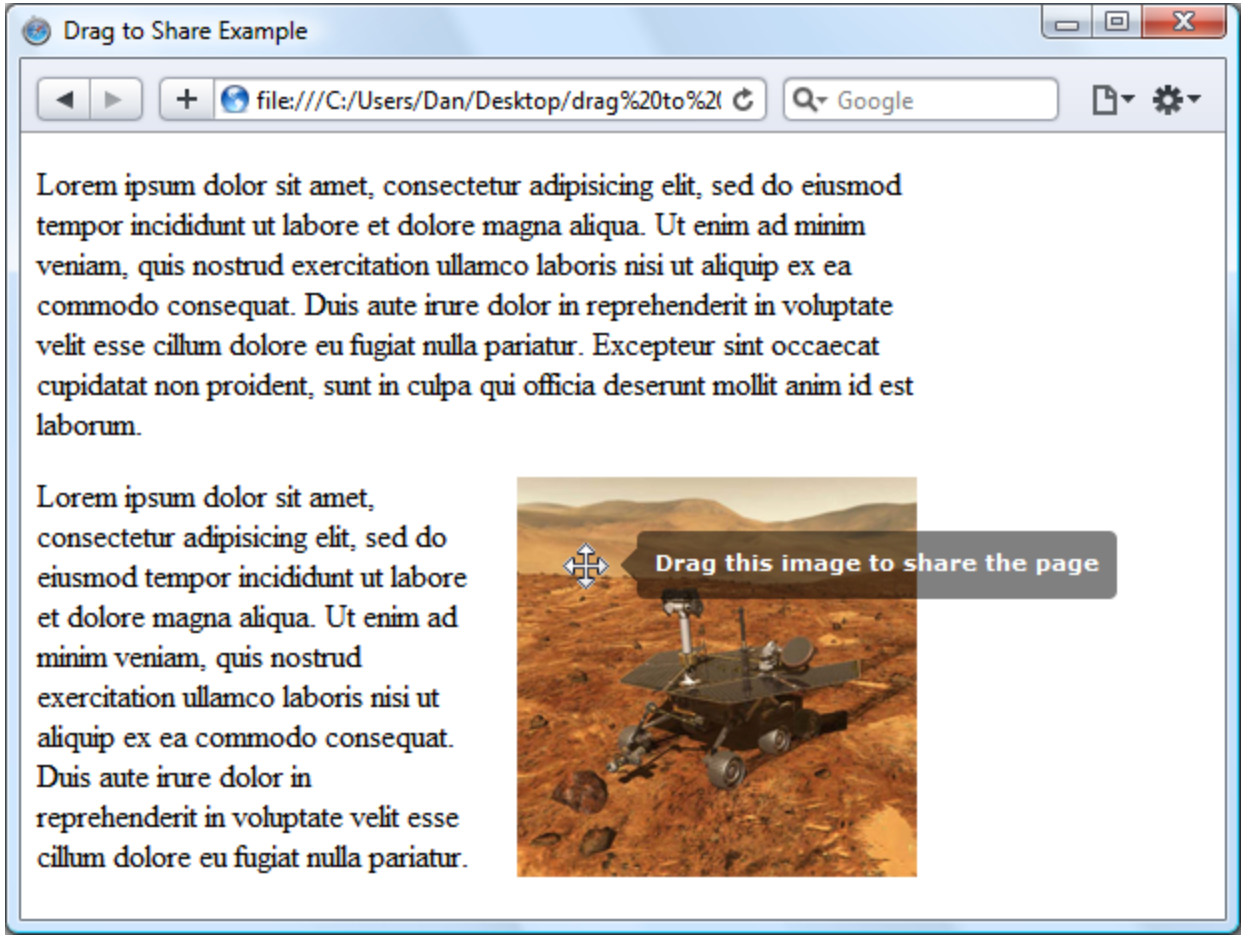
```
#tip {
  position:absolute; display:none; height:25px; padding:9px 9px 0px;
  color:#fff; font-family:Verdana, Arial, Helvetica, sans-serif;
  font-size:11px; font-weight:bold; border-radius:4px;
  -moz-border-radius:4px; -webkit-border-radius:4px;
  background:#000; background:rgba(0,0,0,.5);
}
```

```
#tip .arrow {
  width:0; height:0; line-height:0; border-right:8px solid #000;
  border-right:8px solid rgba(0,0,0,.5); border-top:8px solid transparent;
  border-bottom:8px solid transparent; position:absolute; left:-8px;
  top:9px;
}
```

That's all we need. Most of the styles are pretty basic but we use the border-radius styles to give the tooltip rounded corners in gecko and webkit browsers, and as I mentioned before we use rgba to make the tooltip semi-transparent. This is a great effect and while it's only supported in a few of the common browsers, it's much more efficient than using an alpha-transparent PNG.

The empty span that we added to the tooltip is styled so that it looks like a speech-bubble arrow pointing to the mouse pointer. This is created using the CSS-shapes technique and as it's not CSS3 it's supported in most browsers. It's even supported in IE6, although the border transparency that we give it isn't supported. We could fix this easily enough if we really wanted to, but for the purposes of this tutorial I'm not going to deviate to this topic.

Note that we also use rgba for the border-color of our span so that it blends in with the rest of the tooltip. Now, IE (any version) is not going to support these rgba styles, but as we have provided normal colors before the rgba declarations, in both the tip and the span, the tooltip will just appear solid black in IE. Here's how our tooltip will appear at its best, beautiful isn't it?



Adding the Drop Targets

Ok, so our visitors now know that they can drag the image somewhere in order to share the page, but where do they drag it to? And where can they share it? We now need to react to the image being dragged and show the drop targets, which will consist of a series of links to social networking sites. We've several things we need to do here; we need to add an overlay to the page and create the drop targets first of all.

We could create the drop targets entirely from scratch, on the fly, with jQuery like we did with the tooltip, but out in the wild this would probably result in an unacceptable delay for some visitors. We can minimise this by adding the underlying mark-up for the drop targets to the page, and just showing it when we need to. Directly before the <script> elements at the bottom of the page, add the following code:

view plaincopy to clipboardprint?

```
<ul id="targets">
  <li id="twitter"><a href="http://twitter.com" onclick="javascript:_gaq.push(['_trackEvent','outbound-article','twitter.com']);"></a></li>
  <li id="delicious"><a href="http://delicious.com" onclick="javascript:_gaq.push(['_trackEvent','outbound-article','delicious.com']);">
  </a></li>
  <li id="facebook"><a href="http://www.facebook.com" onclick="javascript:_gaq.push(['_trackEvent','outbound-
  article','www.facebook.com']);"></a></li>
</ul>
```

We use a simple unordered list as the container for the drop targets where each item corresponds to a single target; I've included the social networking sites that I have accounts for, feel free to add more if you wish. Each list item contains a link with its href set to the home page of the social networking site that it represents. We'll need to modify these URLs later in our script, and while more information could be provided in these links, I thought it cleaner to just provide the basic URL in the mark-up.

We need to style these now as well; add the following new styles to the bottom of our style sheet:

view plaincopy to clipboardprint?

```
#targets {
  display:none; list-style-type:none; position:absolute; top:10px;
  z-index:99999;
}
#targets li {
  float:left; margin-right:20px; display:block; width:60px; height:60px;
  background:url(iconSprite.png) no-repeat 0 0; position:relative;
}
#targets li#delicious { background-position:0 -60px; }
#targets li#facebook { background-position:0 -120px; }
```

The outer list container is initially hidden from view so that we can show it when a drag begins. We disable the default icon for list items (a small circle) and position it absolutely at the top of the page. As we want the icons to be visible above the overlay, we set a high z-index on it.

The list items are floated so that they stack up next to each other horizontally and are spaced out with a little margin. The images are set on the list items so an appropriate size is set for them. We're using a sprite file for the images so we need to set the background position of the delicious and Facebook icons.

Note: the icons used in this example came from the Social.me icon pack by jwloh and are available from here. They aren't visible yet, but with the CSS we just added they should look something like this:

Wiring up the Drag Behavior

At this point, we've got out drop targets in place ready to be shown so we now need to set some of draggable's configuration options in order to add the overlay and show the drop targets when a drag begins. We can also create a helper element that will be dragged instead of the underlying <img>. Change the draggable() method so that it appears as follows:

view plaincopy to clipboardprint?

```
1. //make images draggable
```

```
images.draggable({
  //create draggable helper
  helper: function() {
    return $("<div>").attr("id", "helper").html("<span>" + title + "</span><img id='thumb' src='" + $(this).attr("src") + "'>").appendTo("body");
  },
  cursor: "pointer",
  cursorAt: { left: -10, top: 20 },
  zIndex: 99999,
  //show overlay and targets
  start: function() {
    $("<div>").attr("id", "overlay").css("opacity", 0.7).appendTo("body");
    $("#tip").remove();
    $(this).unbind("mouseenter");
    $("#targets").css("left", ($("body").width() / 2) - $("#targets").width() / 2).slideDown();
  },
  //remove targets and overlay
  stop: function() {
    $("#targets").slideUp();
    $(".share", "#targets").remove();
    $("#overlay").remove();
    $(this).bind("mouseenter", createTip);
  }
});
```

We've added an object literal as an argument to the draggable() method containing a series of configuration options; let's look at each option in detail:

helper

We supply an anonymous function as the value of the helper option; the function must return an element, so we create a new div element, give it an id and insert a string of raw HTML. The HTML we insert creates a new span element containing the title of the page. We also create a new image. The image will act as a thumbnail; again we give it an id for styling purposes and set its src attribute to the src of the original image.

cursor

We set the cursor option to pointer so that an action pointer is shown while the drag is in progress.

cursorAt

We use another object literal with the cursorAt option to position where on the drag helper the icon appears. We set the left and top properties of the object to the pixel values. The values we supply set the cursor so that it appears -10px to the left of the helper, and 20 pixels into it.

zIndex

We set the z-index of the helper element using the zIndex option, which sets the style directly on the element as an inline style. This forces the helper above the overlay, and is required because the overlay is inserted after the helper.

start

This option is one of the draggable component's built-in event handlers; whenever a drag interaction starts the function we supply as a value to this option is executed. Within this function we create the overlay and append it to the page, remove the tooltip and stop the current image displaying it again, and then position the targets in the center of the viewport before showing them with a nice slideDown animation.

stop

The stop option is another of draggable's built-in event handlers and is called when the drag stops. All we do here is tidy up, removing the overlay and any text that has been added, sliding up the targets, and rebinding the createTip function to the mouseenter event.

So with these options configured, when we begin dragging the image several things will happen; first of all the overlay will be displayed which will grey out the rest of the page. We also remove the tooltip and prevent it from displayed again if the mouse pointer enters the image again. In most browsers this isn't a problem – the pointer isn't considered 'over' the image because it's on top of the drag helper. But IE (even version 8) does consider it over the image still and displays both the helper and the tooltip.

Our helper element, which will be dragged instead of the original image, will also be created and displayed, showing the page title as a label, and a thumbnail version of the original image. The drop targets will also be shown with a nice animation.

At this stage, we need to provide a little CSS for the helper element, thumbnail and the overlay; add the following selectors and rules to the bottom

of dragToShare.css:

view plaincopy to clipboardprint?

```
#overlay {
  background-color:#000; position:absolute; top:0; left:0; width:100%;
  height:100%; z-index:99997;
}
#helper {
  background-color:#c2c2c2; position:absolute; height:35px;
  padding:15px 70px 0 20px; color:#fff;
  font-family:Verdana, Arial, Helvetica, sans-serif; font-weight:bold;
  font-size:18px; border-radius:8px; -moz-border-radius:8px;
  -webkit-border-radius:8px; border:3px solid #7d7d7d;
}
#thumb {
  width:50px; height:50px; position:absolute; rightright:0; top:0;
  border-left:3px solid #7d7d7d;
}
```

For the overlay we set the background color to solid black, but remember we use jQuery to make it transparent. It's positioned and sized to cover the whole page and has its z-index set to just under that of the drag helper and targets.

The helper element has a range of styles set on it to make it presentable. Mostly these are simple everyday styles that don't warrant discussion, but we do set the border-radius styles here like we did with the tooltip. The height of the helper is fixed, but its width is not, so it will grow or shrink to accommodate all of the alt text from the image that was dragged.

The thumbnail version of the original image is sized and positioned at the far right of the helper. Enough right-padding has been given to the helper element to ensure that the text doesn't run in to the image. When a drag begins, the page should appear like this:

Making the Targets Droppable

Our final task is to make the target social networking icons to react to having the helper element dropped on to them. We do this using the droppable component of jQuery UI, which we selected when we built our jQuery UI download.

Directly after the draggable method (containing the code we just added) add the droppable method:

view plaincopy to clipboardprint?

```
1.  //make targets droppable
2.  $("#targets li").droppable({
3.    tolerance: "pointer",
4.    //show info when over target
5.    over: function() {
6.      $(".share", "#targets").remove();
7.      $("<span>").addClass("share").text("Share on " + $(this).attr("id")).addClass("active").appendTo($(this)).fadeIn();
8.    },
9.    drop: function() {
10.     var id = $(this).attr("id"),
11.       currentUrl = window.location.href,
12.     baseUrl = $(this).find("a").attr("href");
13.     if (id.indexOf("twitter") != -1) {
14.       window.location.href = baseUrl + "/home?status=" + title + ": " + currentUrl;
15.     } else if (id.indexOf("delicious") != -1) {
16.       window.location.href = baseUrl + "/save?url=" + currentUrl + "&title=" + title;
17.     } else if (id.indexOf("facebook") != -1) {
18.       window.location.href = baseUrl + "/sharer.php?u=" + currentUrl + "&t=" + title;
19.     }
20.   }
21. });
```

Again, we use several different configuration options to tailor the implementation to our needs. The options we use are listed below:

tolerance

The tolerance option is used to set when the drag helper is considered to be over a drop target. We've used pointer here so the mouse pointer itself must be over the drop target.

over

We use the over event callback to execute code whenever the drag helper is over a drop target; in this function we first ensure that the element we're about to create is removed, this is to hide elements that may have been created in a previous drag. We then create a new span element and

set its innerText to a message indicating which social network the icon represents.

drop

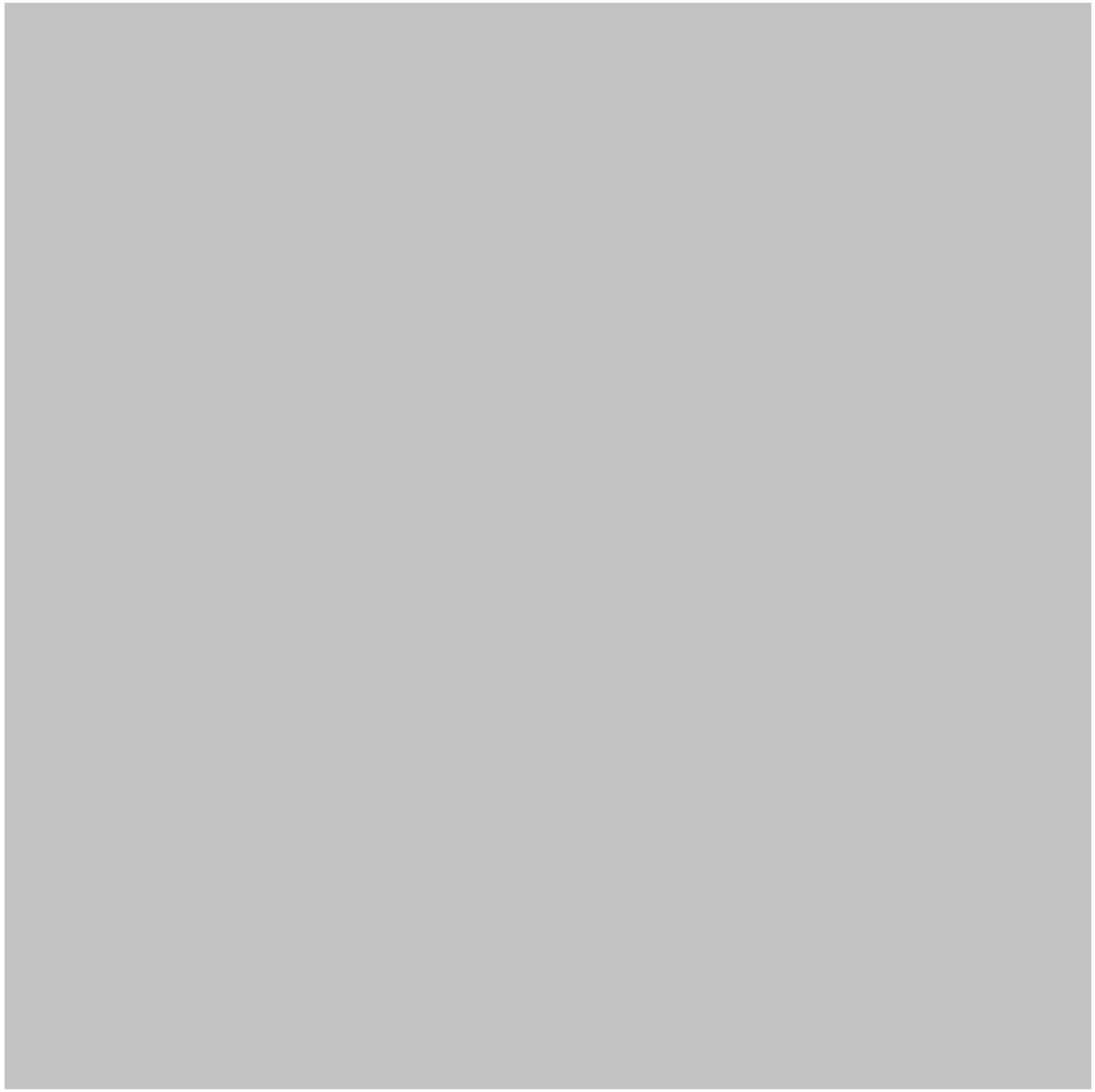
The drop event callback function is where we actually perform the sharing of the page on whichever social network the visitor dropped the drag helper on. We get the URL of the current page and the URL for the social network from the href of the link in the list item representing the icon. Each of the social networks used in this example are able to accept information via the URL, so for example with Twitter, we can add the text for a status update to the input field on the visitor's Twitter page, making it easy for the visitor to share the page title and URL. The information to do this is passed in the URL, so starting with the baseUrl we can build the required URL and then navigate to it with the window.location.href property.

We need one line of CSS and then we're done; to style the share message that we append to the page when the drag helper is over a drop target add the following code:

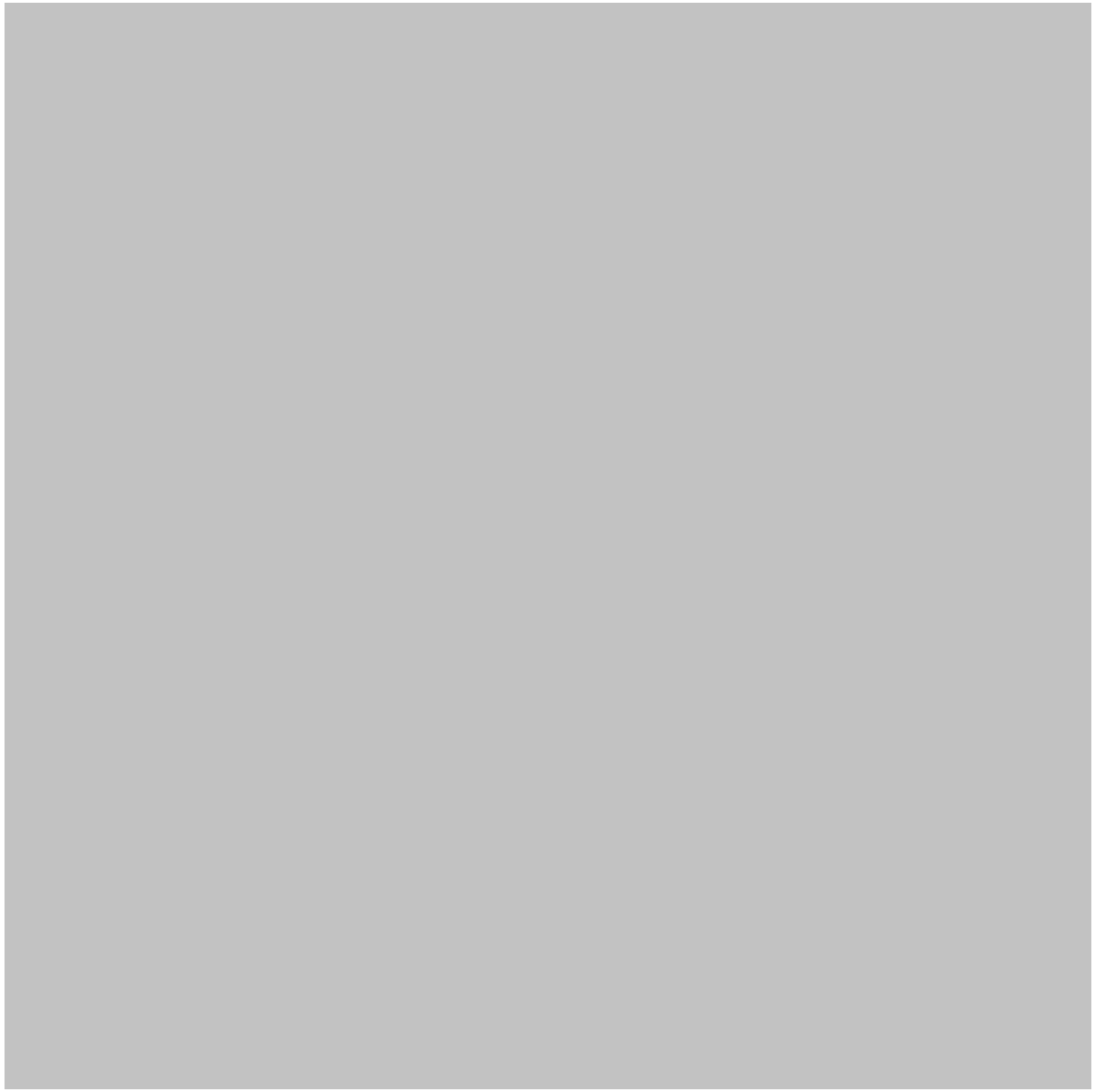
view plaincopy to clipboardprint?

```
1. .share {
2.   font-weight:bold; position:absolute; font-size:14px;
3.   font-family:Verdana; margin-left:-38px;
4. }
```

We should now find that when we drop the drag helper onto one of the icons, the corresponding page should load with the URL of our page displayed:



Summary



Our work here is done; we can now deploy the code to our web pages to provide an easy way for our visitors to share our content across their social networks. This method is good because it's easy to implement consisting entirely of client-side code. We don't need to worry about authentication or anything like that – the visitor will just be prompted to enter their username and password when the external site loads up.

- Follow us on Twitter, or subscribe to the Nettuts+ RSS Feed for more daily web development tuts and articles.

Tags: jQuery

By Dan Wellman

Dan Wellman has been writing web-design and scripting tutorials for approximately 8 years. His new book, jQuery UI 1.8: The User Interface library for jQuery, was released in August 2011.

Note: Want to add some source code? Type <pre><code> before it and </code></pre> after it. Find out more