# An Image Viewer with the DMXzone Universal CSS Transforms Library

The development team at DMXzone have recently produced a stunning jQuery Library that ports cool CSS transforms to all common browsers, allowing us to implement the latest CSS techniques and create stunning animations with ease and which are completely cross-browser compatible.

All of the new CSS transforms are made possible with the Universal CSS Transforms Library including:

rotate
skew (including skewX and skewY)
scale (including scaleX and scaleY)
translate (including translateX and translate)
matrix

The Universal CSS Transform Library is available as open source on GitHub so everybody can use it and contribute.

The library allows us to both get and set any of these CSS transform functions , and allows us to animate most of them. The matrix function cannot be animated at present, and sometimes when animating we are restricted in the functions we can use. The transforms are applied (or obtained) using jQuery's **css()** method which is really handy as it allows us to use a familiar interface to do new and exciting things. Let's look at a basic example to get started with.

With a drop-down menu, it's useful to have an image attached to the top-level item to indicate that it contains a sub-menu. When the menu is opened it's customary to swap the indicator with another to show that the menu is open. With the transforms Library however, we don't need to swap the image. We could do this with the following underlying HTML:

```
<!DOCTYPE HTML>
<html>
<head>
      <meta charset="utf-8">
      <title>DMXzone Universal CSS Transforms</title>
    <link rel="stylesheet" href="css/css-transforms-dropdown.css">
</head>
<body>
      <ul>
            <li>
                  <a id="show" href="#" title="Show Tools">Tools <span>Open</span></a>
                  <ul>
                        <li>
                              <a href="#" title="Image Viewer">Image Viewer</a>
                        </li>
                        <li>
                              <a href="#" title="Code Editor">Code Editor</a>
                        </li>
                        <li>
                              <a href="#" title="HTML Vaildator">HTML Vaildator</a>
                        </li>
                        <li>
                              <a href="#" title="Media Player">Media Player</a>
                        </li>
                  </ul>
```

```
            </li>
      </ul>
      <script src="js/jquery.js"></script>
    <script src="js/jquery.easing.1.3.js"></script>
    <script src="js/jquery.csstransform.pack.js"></script>
</body>
</html>
```

The HTML is pretty standard; we use the HTML5 doctype (although this isn't strictly necessary), link to a style sheet and link to the required script files. The mark-up for the slide out menu is based around a simple nested list structure, with the indicator element defined as a . We could also use the following CSS for this example:

```
ul {
      width:260px; padding:10px; position:relative; float:left; list-style-type:none;
      font:14px Verdana, Sans-serif; background-color:#ccc;
}
ul li { float:left; }
ul a {
      display:block; padding:4px 28px 4px 20px; border:1px solid #aaa;
      border-top-width:2px; border-right-width:2px; position:relative; z-index:999;
      font-weight:bold; text-decoration:none; color:#000;
}
ul a:hover, ul a:active, ul a:focus {
      background-color:#eee; outline:0; border-bottom:1px solid #eee;
}
a span {
      display:block; width:8px; height:8px; position:absolute; right:9px; top:10px;
      text-indent:-5000px; overflow:hidden; background:url(../img/indicator.png);
}

ul ul {
      display:none; width:auto; border:1px solid #aaa; padding:0 0 10px 0;
      position:absolute; left:10px; top:36px; font-size:12px; z-index:0;
      font-weight:normal; background-color:#eee;
}
ul ul li { float:none; }
ul ul a { font-weight:normal; border:none; }
ul ul a:hover { text-decoration:underline; margin-bottom:0; border-bottom:none; }
```

Again, most of this is pretty arbitrary and used purely for the purpose of the example. Finally, we could bring it all together with this simple JavaScript:

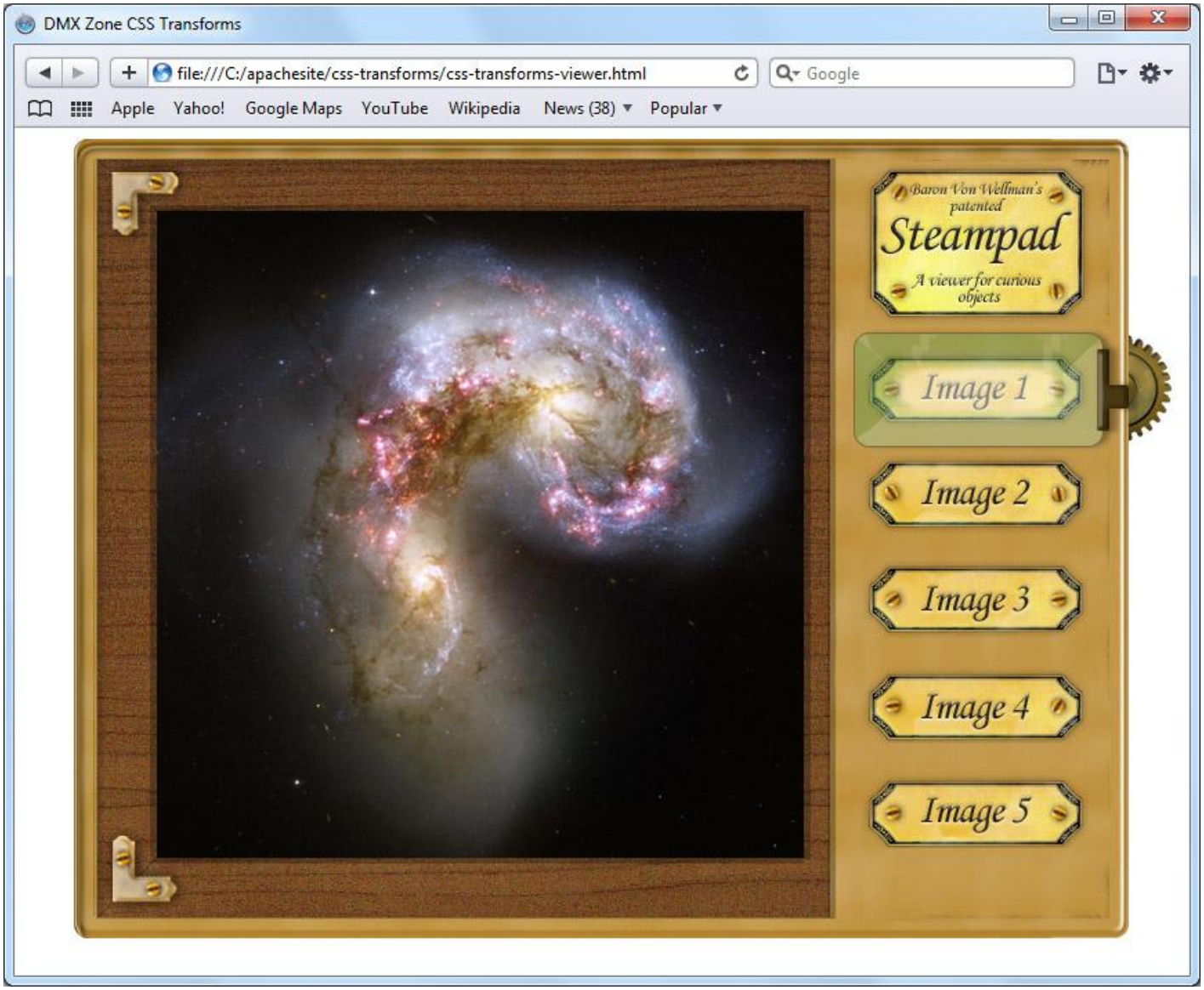
```
<script>
      (function($){

            $("#show").click(function(e) {
                  e.preventDefault();

                  $(this).next().slideDown();
                  $(this).find("span").animate({"rotate": 90});
            });

      })(jQuery);
</script>
```

As you can see, we simply use the DMXzone transform Library to rotate the indicator image once the animation to open the menu has begun. We specify the rotate function as the style property to animate and supply the integer 90 to represent 90 degrees. Try it out; it's a nice effect that requires very little code.
Now let's move on to the main example and use some more of the Library functions. Create a new HTML file and call it **css-transforms-viewer.html**. I decided to use a Steampunk theme for the image viewer, to make the example more fun; the finished page will look like this:



## The underlying page

The HTML for the viewer should be as follows:

```
<!DOCTYPE HTML>
<html>
    <head>
        <meta charset="utf-8">
        <title>DMXzone CSS Transforms</title>
        <link rel="stylesheet" href="css/css-transforms-viewer.css">
    </head>
    <body>
      <div id="viewer">
            <div id="cog"><!-- --></div>
            <div id="window">
                <img id="image1" src="img/image1.jpg" alt="Image 1">
                <img id="image2" src="img/image2.jpg" alt="Image 2">
                <img id="image3" src="img/image3.jpg" alt="Image 3">
                <img id="image4" src="img/image4.jpg" alt="Image 4">
                <img id="image5" src="img/image5.jpg" alt="Image 5">
            </div>
            <div id="ui">
                  <ul>
                        <li id="image1">
                              <a href="#image1" title="Veiw Image 1">Image 1</a>
                        </li>
                        <li id="image2">
                              <a href="#image2" title="Veiw Image 2">Image 2</a>
                        </li>
                        <li id="image3">
                              <a href="#image3" title="Veiw Image 3">Image 3</a>
                        </li>
                        <li id="image4">
                              <a href="#image4" title="Veiw Image 4">Image 4</a>
                        </li>
                        <li id="image5">
                              <a href="#image5" title="Veiw Image 5">Image 5</a>
                        </li>
                  </ul>
                  <div><!-- --></div>
            </div>
        </div>
        <script src="js/jquery.js"></script>
        <script src="js/jquery.easing.1.3.js"></script>
        <script src="js/jquery.csstransform.pack.js"></script>
        <script>
            (function($){

                  //js to go here

            })(jQuery);
      </script>
    </body>
</html>
```

Save the page as **css-transforms-viewer.html**. Let's walk through the structure; the whole widget is encased within an outer container, then within this we have two ** ** elements; the first is the area in which the images will be displayed, the second is the area for the UI which will allow visitors to select an image to view. We also have an empty ** ** that will be styled to appear like a cog, and after the UI elements we have an empty, un-attributed ** ** that we'll use for the over state. At the end of the page we have our required scripts and an awaiting closure into which we can put our code.

## Styling the page

Now we can add the style sheet required for this example; in a new page in your text editor add the following CSS:

```
#viewer {
      width:763px; height:578px; margin:auto; position:relative;
      background:url(../img/viewer.jpg) no-repeat;
}
#window {
      width:468px; height:468px; margin:52px 0 0 60px; position:relative;
      float:left; overflow:hidden;
}
#ui { width:151px; margin:96px 36px 0 0; float:right; }
#ui ul { padding:0; margin:0; list-style-type:none; }
#ui li {
      height:42px; margin-bottom:44px; background-image:url(../img/labels.png);
}
#ui li a {
      display:block; height:100%; overflow:hidden; text-indent:-5000px;
}
#image2 { background-position:0 -43px; }
#image3 { background-position:0 -86px; }
#image4 { background-position:0 -130px; }
#image5 { background-position:0 -174px; }
#ui span { display:none; }
#cog {
      width:79px; height:78px; position:absolute; top:141px; right:-32px; z-index:-1;
      background:url(../img/cog.png) no-repeat 0 0;
}
#over {
      width:208px; height:89px; position:absolute; top:137px; right:-8px;
      background:url(../img/over.png) no-repeat 0 0;
}
```

Save this file as **css-transforms-viewer.css** in the **css** folder. This is all pretty straight-forward; we size up the collection of elements and position them according to the design, as well as setting some background images where appropriate. The most important thing is that the **#window** element is sized to the same dimensions as an individual image, and that it is set to **overflow:hidden** so that the images, which naturally stack up on top of each other, are not all visible, only a single image is visible at any one time.

At this point we should take stock of how the page already behaves, without us having added any additional functionality at all; we can already click on any of the labels and that will cause the corresponding image to be displayed, without any JavaScript at all, which makes the page perfectly accessible to any browser that has JS disabled. Additionally, when a label is clicked, the **href** of the image is added to the URL of the page, which means that the page can be loaded and have a specific image displayed by default, which is quite useful. This functionality works in all popular browsers.

## Adding the JavaScript

To make the example work with our CSS transitions we need to add some script; at the bottom of the page, where we left the comment, add the following code:

```
$("#ui").find("div").attr("id", "over");

$("#window").wrapInner("<div id=\"wrapper\">");

var
imgPositions = { image1: 0, image2: -472, image3: -945, image4: -1417, image5: -1890 },
overPositions = { image1: 0, image2: 75, image3: 152, image4: 230, image5: 310 },
cogPositions = { image1: 5, image2: 80, image3: 154, image4: 235, image5: 310 },
previousCogPosition = 0;
```

We first have to do a little prep work; we give our un-attributed ** ** and id so that it picks up the styling – we do this with JavaScript because the over state is useless if JS is disabled, so it should only be visible is JS is available. We also wrap all of the images within the window part of the viewer in a wrapper ** ** so that we only have to animate one element instead of each individual image. Finally we create an object that is used to store the positions of the images, the cog and the over state so that we can move all of these elements with ease by referring to the object to set their position later in the script. We also define a **previousCogPosition** variable that we'll use to determine how the cog should be rotated.

Next we can define a function that will handle the actual animations:

```
function animator(pointer, notImg) {

      if (!notImg) {
            $("#wrapper").animate({
                  "translateY": parseInt(imgPositions[pointer])
            });
      }

      $("#cog").animate({
            "translateY": parseInt(cogPositions[pointer]),
            "rotate": (parseInt(cogPositions[pointer]) < previousCogPosition) ? "-=365"
: "+=365"
      }, function() {
            previousCogPosition = cogPositions[pointer];
      });

      $("#over").animate({
            "translateY": parseInt(overPositions[pointer])
      });
}
```

This function will receive several arguments, a pointer that tells the animation functions which values to retrieve from the position objects, and a switch that tells the function whether to animate the main images in the window or not. If the **notImg** argument is omitted (or if false is supplied) the image wrapper will not be animated, but the over element and cog still will be. This is necessary in case someone accesses the page the viewer runs on with a hash in the URL, e.g. **viewer.html#image3** – when this happens, the third image will be displayed by the browser automatically , so the over state and cog should be moved so that they are in the correct positions.

The elements are animated using the **translateY** function to alter their position, and additionally the cog will also spin using the **rotate** function. Notice that we check where the cog was previously by looking at our **previousCogPositon** variable (we'll populate this shortly, but it is given an initial value of 0) so that we can rotate the cog the right way – if the cog moves down it will rotate clockwise, if it moves up it rotate anti-clockwise.

We now need to actually check for whether the URL contains a hash and make some additional changes if it is detected:

```
if (window.location.hash) {

      var hash = window.location.hash.split("#")[1], target = $("a[href=" + hash +
"]");

      if (hash === "image2") {
            imgPositions = { image1: 472, image2: 0, image3: -472, image4: -945, image5:
-1417 }
      } else if (hash === "image3") {
            imgPositions = { image1: 945, image2: 472, image3: 0, image4: -472, image5:
-945 }
      } else if (hash === "image4") {
            imgPositions = { image1: 1417, image2: 945, image3: 472, image4: 0, image5:
-472 }
      } else if (hash === "image5") {
            imgPositions = { image1: 1890, image2: 1417, image3: 945, image4: 472,
image5: 0 }
      }

      animator(hash, true);
}
```

If the hash is detected we need to rewrite one of our position objects so that the correct image is displayed when its corresponding link is clicked. The **translateY** function will always detect the wrapper as starting out at 0, even if the last image is displayed on page load (such as when a hash is used in the URI to access the page), so rewriting the **imgPositions** object ensures the viewer still works as intended.

Finally, we just need to wire up the links in out UI structure so that when one is clicked the correct image is displayed:

```
$("#ui a").click(function(e) {
      e.preventDefault();

      var pointer = $(this).attr("href").split("#")[1];

      animator(pointer);
});
```

We simply stop the default behaviour (i.e. to focus the correct image) and call our animator function. The pointer is obtained by splitting the **href** attribute of the link that was clicked. This is then passed into the animator function.

## Summary

In this example we created a steampunked image viewer that uses the DMXzones Universal CSS transforms Library along with jQuery so display a series of images with nice transitions. Using the transforms Library is very easy and is achieved using jQuery's **css** or **animate** methods.

One thing you should be aware of is that using the Library in conjunction with transparent PNGs in IE can cause problems (as you may have noticed in the demo). The Library uses proprietary IE filters in order to make the CSS transitions available to IE and this unfortunately destroys the alpha channel in a transparent PNG. The situation can be mitigated slightly using a program like PNGOptimizer, which removes the alpha channel, although when the transitions are applied, the faulty alpha channel returns.

So go to GitHub and get your own copy of the Universal CSS Transforms Library and enjoy it!