ActionScript to JavaScript:

Finding your feet

Extend your skill set by applying your knowledge of ActionScript to JavaScript and HTML5.

What you will learn...

- How to apply your knowledge of ActionScript to JavaScript
- JavaScript libraries that will simplify development
- Recommended development tools and IDEs
- Efficient methods of animating objects

What you should know...

- Familiarity with ActionScript. Preferably ActionScript 3.0
- HTML and the Document Object Model

Like many Flash developers I've found myself (re) turning to the world of JavaScript – or perhaps you could say "HTML5" development.

This article contains some selected tricks to help adapt your ActionScript knowledge to further your JavaScript skills. There are many ways of using JavaScript, and many libraries to help you out, so it's not so bad if you approach it in the right way.

Development Environments

Let's take a look at editors/development environments first.

This is a subjective area; it's fair to say it's a mixed bag out there. Of all the editors/IDE's out there, my personal vote goes to Netbeans – especially for larger projects. It offers:

- Auto completion, including jQuery
- Eclipse style documentation for functions
- Function and field outlining
- Function and code folding/collapsing
- Refactoring
- Great HTML/5 and CSS support
- JSLint (as a plugin)

Best of all, like FlashDevelop, it's free.

Some others you might consider are:

- Sublime Text
- Aptana Studio
- Dreamweaver

There are also plenty of upcoming browser based editing environments, not least of which is JSBin, built by Remy Sharp and the very promising Brackets by Adobe.

I'm not sure any of them hit the same fine notes as FlashDevelop, which was and is my favourite editor for ActionScript3 projects, but I'm sure things will get better.

Enough about tools though. Let's focus on the language.

Why JavaScript?

HTML5 is a hot topic at the moment, but let's face it, most of the good stuff is accessed through JavaScript. Indeed, there are some stunning JavaScript examples out there, as people are squeezing the best from the language/browsers.

It's been interesting to see how JavaScript has changed its 'image' over the last few years. From being associated with ugly effects and tiresome popup windows, now it is key to getting the most out of "HTML5" and associated APIs, and the browser itself has become an increasingly powerful programming environment.

Setting up…

Let's start at the beginning

The first things you'll want to add to your scripts are some very useful JavaScript libraries that make life easier scripting across the various browsers out there. I'll quickly mention a few utilities that I use commonly…. You can use templates such as HTML5 BoilerPlate, which is a "template for a fast, robust and future-safe site". *http://html5boilerplate.com* It includes much of the following utilities.

Modernizr

Modernizr contains some handy scripts that can check for the HTML5 ability of any browser, as well as a few other goodies. I'd recommend it's one of the first scripts you add to a page:

```
<script src="http://cdnjs.cloudflare.com/ajax/libs/
modernizr/2.5.3/modernizr.min.js"></script>
```

Once added, you can easily check for features like so:

```
var isCanvasAvailable = Modernizr.canvas;
//this will return a boolean value
```

jQuery

jQuery is almost ubiquitous these days. It's described as a "write less, do more" framework and it is indeed very useful for speeding up the workflow. Admittedly, in the wrong hands you can end up with nested functions within nested functions. That said, jQuery is great, and can make life much easier across browsers. Along with Modernizr, I'd recommend it's the first thing you add in your header as a default.

There's enough documentation elsewhere on jQuery but suffice it to say it comes into its own when you need to access the DOM, animate elements and create callbacks for events.

Add it to the page like so:

```
<script src="http://ajax.googleapis.com/ajax/libs/
jquery/1/jquery.min.js"></script>
```

Underscore

Underscore is a "utility-belt library" with a suite of functions for JavaScript – it can make life much easier scripting for the many different browsers and it's intended for functional programming:

```
<script src="http://documentcloud.github.com/
underscore/underscore-min.js"></script>
```

Making ourselves a little more at home…

Here's an easy but very useful utility, which utilises underscore.js, and that you can use to make yourself feel

ADV

more at home: a `trace` function for logging and debugging. I was shown this trick by another developer I worked with, and I've found it very useful. You can add this to your own utilities functions file if you wish.

```
var trace = function (obj) {
  if (!_.isUndefined(window.console) &&
     !_.isUndefined(window.console.log)) {
    window.console.log(obj);
  }
}
```

The `console.log()` method is the browser/JavaScript equivalent of trace. It'll write out strings and objects to the debugging console. The problem is, it's not all that consistent across browsers.

Now we can at least consistently write to the console, and using the familiar `trace()` method. It all helps…!

Simply call it like you would in ActionScript:

```
trace("hello"); //trace a string
trace(myObject); //trace an object
```

ActionScript coding techniques mapped to JavaScript

ActionScript and JavaScript share an ECMAScript heritage, so there are indeed ways of scripting like ActionScript in JavaScript. Contrary to popular belief it is possible to create classes, public/private variables and even constructors once you get to grips with JavaScript. Although it will feel "quirky", to say the least…

Listing 1. *xxxxxxx*

```
Particle = function(x,y,0xFF0000) {
  // Default to 0 if parameter(s) isn't defined
  this.x = x || 0;
  this.y = y || 0;

  trace("Particle created at x: " + this.x +
    " and y: " + this.y);

  var _name = "particleName";

}

// create instance of Particle
var particle = new Particle(100,100, 0xFF0000);
var blueParticle = new Particle(100,100, 0x0000FF);

trace(particle.colour);
trace(blueParticle.x);
```

Classes and Object Oriented JavaScript

Defining classes is as easy as defining functions, because it is practically the same thing. After creating a function, you can make instances of it. The example below creates a `Particle` class with public variables called `x` and `y`. After that we create two instances of the `Particle` class with some other values.

Note

I use the same code conventions as Actionscript: Classes should be UpperCased, functions/variables should be lowerCased and constants should be ALL_CAPS. Also I sometimes use an underscore before private variables.

Here's an example `Particle` "class", which has private variables and public functions (Listing 1).

By using the `this` keyword we're essentially saying for example that `this.x` is a public variable.

To make a variable private we would limit it to the scope of the function by declaring it like so:

```
var name
```

Listing 2. *xxxxxxxxxxxxxx*

```
Particle = function(x,y)
{
  // Default to 0 if parameter(s) isn't defined
  this.x = x || 0;
  this.y = y || 0;

  trace("Particle created at x: " + this.x +
    " and y: " + this.y);

  function getRandomColour()
  {
    return Math.random * 0xFFFFFF;
  }
}

var Particle = new Particle(100,100, 0xFF0000);
trace(particle.x); // returns x position
trace(particle.getName()); // returns name

// not possible :
// error in console; property does not exist,
                    because function is private.
trace(particle.getRandomColour());
// error in console; property does not exist,
                    because variable is private.
alert(particle._name);
```

Private variables can be declared with the `var` keyword inside an object, and can only be accessed by private functions and privileged methods in the scope. You might also want to add an underscore to easily identify private variables, like so:

```
var  _name
```

This is an example of encapsulation in JavaScript – that is, allowing an object to group both private and public members under a single name, and restricting access to other objects.

Public functions

Here's how you'd go about making a public function within the `Particle` class:

```
this.getName = function(){
     return _name;
}
```

Listing 3. *xxxxxxxxx*

```
function namespace(namespaceString) {
  var parts = namespaceString.split('.'),
parent = window,
    currentPart = '';

  for(var i = 0, length = parts.length; i < length;
                    i++) {
    currentPart = parts[i];
    parent[currentPart] = parent[currentPart] || {};
    parent = parent[currentPart];
  }

  return parent;
}
```

Listing 4. *xxxxxxxxx*

```
var project = namespace('com.mycompany.project');
project.Particle = Particle = function(x,y,0xFF0000)
// create class inside package
{
  this.x = x || 0;
  this.y = y || 0;
  var _name = "particleName";

}
var myParticle =  new project.
                    Particle(100,200,0xff0000);
trace(myParticle.x);
```

This could then be called from an instance:

```
particle.getName();
```

Private Functions

Here's how we'd go about creating a private function (Listing 2).

By using these principles, possibly in combination with namespaces (which we'll look at next) you can create clean JavaScript code.

ActionScript3 style "Packages" in JavaScript using Namespaces

In JavaScript it can be easy to run into conflicts with function and variable names because of unwanted duplicates. Using namespaces can help to prevent those conflicts.

There's no dedicated syntax for namespaces in the language, but we can get the same benefits by creating a single global object and adding all our objects and functions to the object.

You can also use the same naming conventions for namespaces as in ActionScript. Suppose we want to use a namespace like `com.mycompany.project`. We can accomplish this using this approach.

```
var com = com || {};
com.mycompany = {};
com.mycompany.project = {};
```

….or something like this:

```
var com = com || {
  mycompany: {
    project: {}
  }
};
```

Compared to AS3 this is very cumbersome and hard to maintain as your code expands. However, here's another utility function that would let you do the same thing using a single line of code: Listing 3.

You could then create a namespace like this with one line of code: Listing 4.

Using this approach requires less typing and the resulting code also looks less verbose. Some JavaScript libraries, such as Dojo, provide their own namespace utility functions: *http://dojotoolkit.org*.

Now it looks a little more like Actionscript 3, only there are no imports. This works in all browsers.

Type Declaration

You can't declare a variable's data type like we can in ActionScript. This is because JavaScript is loosely typed, so…

```
var myNumber:Number
```

…would simply be written as…

```
var myNumber
```

In ActionScript if you tried to assign it as anything other type than Number it would throw an error, whereas with JavaScript you could go on to use it to store a string. This can be a pain when trying to remember what type of variable is being passed to a function. It can be a good idea to remind yourself what type a variable is with a small comment before the arguments in the function declaration:

```
function (/* object */ options){
  // function logic would go here
}
```

Constants

Javascript has a const keyword too, you could use it instead of var. It declares a constant: a variable whose value can't be changed after it's initialised. However, bear in mind this is not implemented in Internet Explorer, so it is of limited use.

Extending Classes

You can even extend your JavaScript classes. This is how might go about creating a class that extends the `Particle` class. We'll call it `FireParticle`.

```
FireParticle = function(x,y) {

  Particle.call( this, x, y );
  trace("this particle is a fire particle!")

}
```

Calling the super constructor is really easy. All you have to do is use the `call()` method.

In the code above, what we are asking the Javascript engine to do is to execute the `Particle()` method (base constructor) using the `FireParticle` instance as the `this` context. We are then forwarding the arguments to the super constructor for use. You could also use Javascript's `apply()` method to invoke a base constructor and pass in the context in which it will execute.

There's not really that much more involved in extending classes in Javascript, but there's how it's done.

Using prototype

As with earlier versions of ActionScript, the prototype object of JavaScript makes it possible to add custom properties/methods to all instances of an object. You have to reference the keyword `prototype` on the object before adding the custom property to it, and this property is instantly attached to all instances of the object. Let's apply that to our `FireParticle` class: Listing 5.

Listing 5. *xxxxxxxxxxxxx*

```
FireParticle = function(x,y) {
  Particle.call( this, x, y );
  trace("this particle is a fire particle!")
}

FireParticle.prototype = new Particle();
FireParticle.prototype.constructor = FireParticle;
FireParticle.prototype.updatePhysics = function() {
  /* update particle physics here */
};
```

Listing 6. *xxxxxxxxx*

```
//get reference to DOM element
var div = document.getElementById("div1");
//add a MouseClick event
div.addEventListener("click", onClick, false);
function onClick( ev ){
  trace(ev.target);
}
```

Adding event listeners

In JavaScript you can add event listeners just as you would do for objects in AS3: Listing 6.

You can similarly use the `removeEventListener()` method to remove an event listener that has been registered with the `addEventListener()` method.

One *big* drawback with the above code is that Internet Explorer versions older than 9 are not supported and you'd need to use `addEvent()` instead. This is a pain… For a solution that works across browser I tend to use jQuery, with this syntax:

```
$("#div1").bind("click", onClick);

function onClick( ev ){
  trace(ev.target);
}
```

Animating objects with requestAnimationFrame

In ActionScript you will have been using `ENTER_FRAME` events or a `Timer` to animate things. In JavaScript, traditionally you would have used either `setInterval()` or `setTimeout()`, which would look something like this:

```
setInterval( draw, 1000/30); //30 fps
function draw(){
  // draw stuff
}
```

Or this:

```
function draw() {
  setTimeout(draw, 100);
  // Drawing code goes here
}
draw();
```

The problem is that `setTimeout()/setInterval()` doesn't know what else the browser is doing. The page could be minimised or hidden in a tab, using up processing power when it doesn't need to on some browsers.

Also, `using these methods` only updates the screen at a set rate, and if your animation frame rate is not in synchronised with the redrawing of your screen, it could take up more CPU.

Not exactly efficient.

The `requestAnimationFrame()` function, which was proposed by Mozilla and later adopted by the WebKit vendors, provides a native API for running any type of animation in the browser – and that can be for DOM elements, CSS, canvas and so on.

This is the syntax:

```
function draw() {
  requestAnimationFrame(draw);
  // Drawing code goes here
}
draw();
```

Listing 7. *xxxxxxxxx*

```
https://gist.github.com/1579671

// http://paulirish.com/2011/requestanimationframe-for-smart-animating/
// http://my.opera.com/emoller/blog/2011/12/20/requestanimationframe-for-smart-er-animating

// requestAnimationFrame polyfill by Erik Möller
// fixes from Paul Irish and Tino Zijdel

(function() {
  var lastTime = 0;
  var vendors = ['ms', 'moz', 'webkit', 'o'];
  for(var x = 0; x < vendors.length && !window.requestAnimationFrame; ++x) {
    window.requestAnimationFrame = window[vendors[x]+'RequestAnimationFrame'];
    window.cancelAnimationFrame = window[vendors[x]+'CancelAnimationFrame']
    || window[vendors[x]+'CancelRequestAnimationFrame'];
    }

  if (!window.requestAnimationFrame)
    window.requestAnimationFrame = function(callback, element) {
      var currTime = new Date().getTime();
      var timeToCall = Math.max(0, 16 - (currTime - lastTime));
      var id = window.setTimeout(function() { callback(currTime + timeToCall); },
        timeToCall);
      lastTime = currTime + timeToCall;
      return id;
    };

  if (!window.cancelAnimationFrame)
    window.cancelAnimationFrame = function(id) {
      clearTimeout(id);
    };
}());
```

Very much like the `setTimeout()` version but with `requestAnimationFrame()` instead, and no interval rate, so the frame rate is down to the speed of the computer (and the capabilities of the browser). You can also pass an optional parameter along to the function that's being called, such as the current element being animated. The browser is then able to draw when it can, rather than being forced to draw no matter what else it is doing.

It will also group all animations into a single "frame update", which will save processing power, with the idea that this is a much more mobile device friendly method.

The downside to `requestAnimationFrame()`? Its compatibility across browsers varies, and also, given it is a new standard, you also have to deal with "vendor prefixes" – e.g. `webkitRequestAnimationFrame()`. Also, you can forget about it when it comes to Internet Explorer – though it is supported in version 10.

Fortunately Paul Irish, Erik Moller and Tino Zijdel have come to the rescue with a "polyfill" (a term used to describe code snippets/libraries that can make new features work in older browsers) that you can get from here. This script will make `requestAnimationFrame()` run across almost all browsers: Listing 7.

What about controlling the frame rate?

The only thing missing in the above code is how to throttle the frame rate to one of your choosing. Don't worry, there is a way of doing that, and it's by using `setTimeout()` as well. Here's an example that will attempt to draw at 30 frames per second, but with the advantage of `requestAnimationFrame()`, so it will only draw the new frame if it can, and also won't go hogging resources if you're on a different window or tab.

```
function draw() {
  setTimeout(function() {
    requestAnimationFrame(draw);
    // Drawing code goes here
  }, 1000 / 30);
}
```

Drawing with JavaScript

Up until the rise of HTML5 the only way to animate things with JavaScript was through the DOM (Document Object Model), by moving HTML elements around the page. An interesting area of the HTML5 specification for ActionScript developers is the new drawing APIs: Canvas, SVG, and WebGL. These provide bitmap, vector, and three-dimensional drawing capabilities, respectively.

The canvas element in particular will seem familiar to ActionScript developers. It's a 2D drawable region where you can use JavaScript to draw and animate complex graphics and images. Given the close relationship between JavaScript and ActionScript it's not such a huge stretch to port Flash drawing code.

Canvas, SVG, and WebGL would all need a full article to do them justice, but suffice it to say I have found Canvas and WebGL (via the excellent Three.js library) to be very powerful. There are also some really interesting polyfills out there, such as Flash Canvas: *http://flashcanvas.net*, which emulates HTML5 canvas features on earlier versions of Internet Explorer through … you guessed it.... Flash!

In conclusion

So there you go. We've only scratched the surface, but as you can see JavaScript is very powerful if you take the time to learn its intricacies. It's not such a huge step from ActionScript as it turns out, and if you tap into its Object Oriented Programming potential you will certainly glean the best of it, although it's a shame there's no type declaration and that the IDE's aren't up to the same level we're used to with AS3. Equally there are many excellent libraries and polyfills out there to make life easier when it comes to dealing with the different versions of browsers.

The fact is that JavaScript is the only language used natively in the browser environment (that does not require a plugin), therefore it's the de facto language of the web. If used properly you'll be able to make very powerful and rich applications, and it's certainly worth embracing this as the web shifts towards mobile devices and browsers.

RICHARD ENGLAND

Richard England is an award winning freelance developer, who has worked with Flash technology for over ten years, building FWA winning sites, BAFTA-nominated games and interactive learning resources.

He runs short HTML5 and JavaScript courses for NTI Leeds, teaching how to build web apps using open web technologies. His Twitter ID is @englandrp and he can be contacted at info@richardengland.co.uk.