baconjs library specification

1.1 Edition / 7th November 2011

Table of Contents

1.00 – Introduction

baconjs will be a lightweight but powerful JavaScript library which aims to make common JavaScript tasks easier and faster. It will also replicate newer features on older browsers (such as the lack of JSON object on Internet Explorer versions less than eight), and will include an advanced plugins system. It will also come with a couple preinstalled plugins, such as a basic testing library that can be used to test the users JavaScript.

1.01 – Coding Standards

baconjs should use as similar coding standards to the JavaScript core as possible. It should use camel-caps in variable names and function names:

```
variableName = functionName(args);
```

It should use single quotes for strings, and have parentheses on the same line:

```
(function() {
	var i = 0, foo = 'bar';
	return {
		addOne: function() {
			return ++i;
		}
	};
})();
```

Tabs should be four spaces wide, and tabs should be used for indents, not spaces. UNIX (LF) line endings should be used.

All code should be readable and well documented.

1.02 – Specification Changelog

1.1 – 2011/11/07:

- Deprecated `.parents` as it was almost identical to `.parent` (3.02).
- Replaced `.get` with `.select`, added `.get`, `.each` and `.matched`. Also added some extra info to the DOM manipulation functions (3.02).
- Added more information to `.live` on how the events should be captured (4.02).
- Added the data helper (7.04).
- Added `.querySelector` and `.querySelectorAll` to the old browsers compatibility section (9.03).
- Added a changelog (1.02).
- Added more detail to the testing library specification (8.01).
- Removed a bad parameter from `Array.max` and `Array.min` (7.03).
- Renamed the badly named “Old JavaScript Compatibility” section to “Old Browsers Compatibility” (9.00).
- Fixed a typo in the opacity example for the animations (6.00).
- Renamed `.show` and `.hide` to `.fadeIn` and `.fadeOut` respectively (6.00).

1.0 – 2011/11/04:

- Added initial specification.

2.00 – Element selection

There should be a function to select an element or a group of elements using CSS-like selectors. This can be very powerful and a lot easier than the traditional method of using `document.getElementById` / `document.getElementsByClassName` / `document.getElementsByTagName`, and should work in all browsers (for example, `document.getElementsByClassName` does not work in older versions of Internet Explorer). The function should not be too resource heavy, and should be compatible with all browsers. The function should be called `$`.

```
object $ ( str selector [ , int limit ] )
```

It should accept two parameters: the selector string by which to select the elements from, and an optional parameter called "limit" - if selecting a collection of elements, this parameter should specify the maximum number of elements to be selected. Any elements selected after this number of elements have been selected should be ignored. It should default selecting an infinite number of elements. The function should return an object containing the elements / group of elements which can then be used for other baconjs functions:

```
$('.jsdisabled').remove();
```

Following are some examples of selectors that could be used:

- `.class_name`
- `#element_id`
- `tag_name`
- `[attr='something']`
- `[attr^='sth']`
- `div.body form[data-ajax] submit`

3.00 – Document Object Model

baconjs should contain a powerful and comprehensive set of functions for working with the Document Object Model (DOM).

3.01 – DOM Traversal

baconjs should contain a set of functions for traversing the Document Object Model. They should aim to use as little resources as possible, and should be fast and easy to use. There should be the following functions:

```
object bacon_el.next ( [ str selector ] )
object bacon_el.prev ( [ str selector ] )
object bacon_el.children ( [ str selector [ , int limit ]] )
object bacon_el.parent ( [ str selector ] )
object bacon_el.siblings ( [ str selector [ , int limit ]] )
object bacon_el.closest ( str selector )
object bacon_el.select ( str selector [ , int limit ] )
```

A couple notes:

- `.parent` should attempt to return the immediate parent, and then if it doesn't match the selector, try the parent of the parent etc. If selector is not specified, then return the immediate parent.
- `.siblings` should return *only* the siblings of the element, and not the element itself.
- The difference between `.children` and `.select` is that `.children` should check only one level while `.select` should check every child element that it has.
- `.select` should attempt to use `.querySelectorAll`, or replicate its functionality if it is not available.

3.02 – DOM Manipulation

baconjs should contain a set of functions for manipulating the Document Object Model. The following format should generally be used, and where possible the function should return `this`.

```
object bacon_el.append ( obj element )
```

Can be either a string of HTML (see 3.03) or an existing object:

```
$('#test').append('<strong>test</strong>');
$('#test').append(test);
```

```
object bacon_el.appendTo ( str selector [ , int limit ] )
object bacon_el.prepend ( obj element )
object bacon_el.prependTo ( str selector [ , int limit ] )
object bacon_el.moveTo ( str selector [ , int limit ] )
object bacon_el.copyTo ( str selector [ , int limit ] )
object bacon_el.insertAfter ( obj element )
object bacon_el.insertBefore ( obj element )
```

The following functions should use the animation functions:

```
object bacon_el.remove ( [ int time [ , func callback ]] )
object bacon_el.hide ( [ int time [ , func callback ]] )
object bacon_el.show ( [ int time [ , func callback ]] )
```

3.03 – Miscellaneous DOM

There are a few more functions that should be included that fit into neither of the above categories:

```
object bacon_el.get ( int number )
object bacon_el.each ( func fn )
object bacon_el.matched ( str selector )
```

3.04 – HTML Parser

As demonstrated in some of the above functions, baconjs should be able to parse HTML and accept that in the place of a real element. For example, the following code sample should return a DOM element, not a string:

```
$('<div><strong>Warning:</strong> Something</div>');
```

4.00 – Event Handling

baconjs should make it easy to handle events across browsers, while maintaining high speeds and low resource usage.

4.01 – `.on` / `.one`

The `.on` and `.one` functions should allow basic event handlers to be added:

```
object bacon_el.on ( str event , func cb [ , bool bubble ] )
object bacon_el.one ( str event , func cb [ , bool bubble ] )
```

The `.on` function should add the handler until either the handler is destroyed using `.removeHandler` or `.off`, or until the element is destroyed. The `.one` function should be similar, but should only call the handler once. The callback should be sent the event information as a parameter, and the element as `this`.

4.02 – `.live`

The `.live` function should be similar to the `.on` function, but should call the handler for all elements that match the selector, now and in the future. There is no equivalent function for `.one`.

```
object bacon_el.live ( str event , func cb [ , bool bubble ] )
```

It should function by capturing all events of that type on the document (by binding to the document itself) and scanning for all elements that match the selector.

4.03 – `.removeHandlers` / `.off`

`.removeHandlers` should remove either all handlers for that element, all handlers of that type for that element, or a specified handler. `.off` should be an alias for `.removeHandlers`, as it is shorter but less descriptive.

```
object bacon_el.removeHandlers ( [ str event [ , func cb ]] )
```

4.04 – `.trigger`

.trigger should trigger the specified event handlers and behaviours for that element. The callback will be triggered when all the handlers have been successfully triggered:

```
object bacon_el.trigger ( str event [ , func callback ] )
```

5.00 – AJAX Helper

The AJAX helper should be similar to jQuery's AJAX helper, and should automatically perform all the trivial tasks, making the performance of AJAX requests very easy for the coder. It should support custom HTTP methods.

```
bool $.req ( str method , str url [ , str data [ , func
callback ]] )
```

The data argument can either be a string (eg `arg1=data&arg2=data`) or an object which will then be converted to a querystring. If the data argument not required, the callback argument can be moved forwards, eg:

```
bool $.req ( str method , str url [ , func callback ] )
```

An object can also be specified as the first object. When using an object as the first argument, all other arguments will be ignored and the following properties of the object will be used:

```
bool $.req ( obj options )
options = {
     method: str method,
     url: str url,
     data: str method (optional),
     callback: func callback (optional),
     error: func error_callback (optional)
}
```

5.01 - Aliases

There will also be aliases, `$.post` and `$.get`. They will accept the same parameters as the `$.req` function, excluding the method argument:

```
bool $.get ( str url [ , str data [ , func callback ]] )
bool $.post ( str url [ , str data [ , func callback ]] )
```

6.00 – Animations

baconjs should contain a comprehensive and powerful animations library that is capable of performing such tasks as fading in or out elements, progressively changing properties, etc.

6.01 – Mechanisms

The animations library should use setInterval to progressively change the property. In this example, we will be fading out an element from opacity 1 to opacity 0 over 1000ms. The difference in opacity (1) should be divided by the time in milliseconds (1000), and then a setInterval at 1ms should change the value. It should round the value to the nearest 0.01 and only change the value if it is any different to the previous value.

6.02 – `.fadeIn` / `.fadeOut`

The `.show` and `.hide` functions should change the opacity of an element to 100% and to 0% respectively.

```
object bacon_el.fadeIn ( [ int time [ , func callback ]] )
object bacon_el.fadeOut ( [ int time [ , func callback ]] )
```

6.03 – `.animate`

The `.animate` function should have the ability to change the value a numeric CSS property. It should also have an `.anim` alias.

```
object bacon_el.anim ( [ int time , ] obj css  [ , func cb ] )
```

The CSS property should be as follows:

```
css = {
     property: int new_value,
     property2: int new_value
}
```

7.00 – Miscellaneous Helpers

There should be a few more, minor, features that are not worth their own major section:

7.01 – Querystring Helper

There should be a basic querystring helper to convert querystrings to and from objects. It should automatically detect whether it is being passed an object or a string and convert it to the other.

```
obj $.querystring ( str querystring )
str $.querystring ( obj querystring )
```

7.02 – Form Helper

There should be a form helper which can perform such tasks as to serialise the form into an array or querystring. This can be used alongside the AJAX helper to AJAXify the form.

```
str bacon_el.serialise ( [ output_array = false ] )
```

7.03 – Array Helper

baconjs should add some useful common features to arrays. They should be added to the Array prototype, but should be easy to disable

```
bool Array.included ( value )
```

Should return true if `value` is contained in the array.

```
int Array.max ( [ str type ] )
int Array.min ( [ str type ] )
```

Should return the maximum and minimum value of an array respectively. Should be able to handle both Numbers and Strings, but not in the same array; if the array contains two different types then it should use either the type specified in the first parameter, or the type of the first item.

```
int Array.range ( )
```

Should return the difference between the minimum value and the maximum value.

```
obj Array.groupBy ( func sortBy )
```

Should sort an array into groups by the sortBy function. For example:

```
[1.5, 2.3, 2.6, 3.3, 3.6].groupBy(function(num) {
     return Math.floor(num);
});
=> {
     1: [1.5],
     2: [2.3, 2.6],
     3: [3.3, 3.6]
}
```

```
array Array.shuffle ( )
```

Should shuffle the array.

```
Array.rand ( [ int num ] )
```

Should return a random element, or random selection of elements, from the specified array. If returning multiple elements, should not return the same element more than once. If `num` is specified as one, it should return an array with only one item in (as opposed to returning just the item if unspecified).

```
array Array.compact ( )
```

Should remove all falsey values from an array.

```
array Array.flatten ( )
```

Should remove all nesting from an array, for example:

```
[1, [2], [3, [[4]]], [5]].flatten();
=> [1, 2, 3, 4, 5]
```

```
array Array.without ( value [ , bool array ] )
```

Should return the array with all instances of `value` removed. If `array` is true, then `value` should be treated as an array of values which will all be removed.

```
array Array.unique ( )
```

Should return the array with all duplicate entries removed.

7.04 – Data Helper

baconjs should contain a way to store data in a given element. It should use the same syntax as jQuery, but should use `.dataset` or create it if it doesn't exist.

`bacon_el``.data( get [ , set ] )`

If `set` is set, then index `get` should be set to `set`, and if `set` isn't set then index `get` should be returned.

8.00 – Plugins

baconjs should contain an advanced plugin system.

To add a function to the baconjs object, it should be added to the `$` object:

```
$.newFunction = function() {};
```

To add a function to a baconjs HTML object, it should be added to the `$.fn` object:

```
$.html.newFunction = function() {};
```

It can then be called by:

```
$(selector).newFunction();
```

8.01 – Testing Library

baconjs should have a simple testing library as a plugin to test JavaScript applications. It should be powerful enough to be able to test anything it is required to test, and should support both synchronous and asynchronous operations. It should be similar to the Jasmine testing library.

8.01.1 – General Syntax

The general syntax should be as follows:

```
$.describe('Group Name', function() {
     $.it('should … (test name)', function() {
          $.expect(variable).toEqual(variable);
     }
});
```

8.01.2 – Groups and Tests

Groups are groups of tests initiated by the `$.describe` function. It should be possible to run only one group at a time by specifying it in the URL (eg `?group=groupid`). Tests are initiated by `$.it`, and it is during initiation that it should be specified whether it is synchronous or asynchronous.

Initiating a group:

```
void $.describe ( str groupName , func groupFunction )
```

Initiating a test:

```
void $.it ( str desc , func testFunc [ , bool async ] )
```

The async parameter should be a boolean value or an integer value specifying the timeout. If it is `true`, the timeout should default to 400ms. If it is `undefined` or `false`, it should be assumed that the test isn't asynchronous. If a test is asynchronous, the test function should be passed a `done` function as the parameter:

```
$.it('should be asynchronous', function(done) {
     setTimeout(function() {
          $.expect(true).toBeTruthy();
          done();
     }, 10);
}, true);
```

If it doesn't return in time, it should timeout and display that as the error.

8.01.3 – Test Methods

It should have the following methods:

```
.toEqual ( match )
```

Should test whether the two values equal each other using double equals. If it is an array or an object, it should test whether the values equal each other.

```
.toNotEqual ( match )
```

Should negate `.toEqual`.

```
.toBe ( value )
```

Should test whether the two values are identical using triple equals. Should return true if they are identical.

    .toNotBe (value)

Should negate `.toBe`.

    .toMatch (regex)

Should return true if the specified regex matches the input.

    .toNotMatch (regex)

Should negate `.toMatch`.

    .toContain (value)

Should return true if `value` is contained in the input. If the input is an array, should see if `value` is one of the entries.

    .toNotContain (value)

Should negate `.toContain`.

    .toBeLessThan (value)

Should return true if the input is less than `value`.

    .toBeGreaterThan (value)

Should return true if the input is greater than `value`.

    .toBeDefined()

Should return true if the input is defined.

    .toBeUndefined()

Should return true if the input is undefined.

    .toBeNull()

Should return true if the input is null.

    .toBeTruthy()

Should return true if the input is truthy.

    .toBeFalsy()

Should return true if the input is falsy.

9.00 – Old Browsers Compatibility

It is important for most websites that they can be accessed by older browsers without the user experience being degraded or the site not working at all. baconjs should attempt to remedy any errors by reproducing any newer functions that don't exist in older browsers.

9.01 – Array.forEach

In JavaScript 1.6, the `.forEach` method was added to the Array object. Older browsers do not support it, so baconjs should add support for it for the browsers which do not support it.

9.02 – JSON

Some older browsers (for example, Internet Explorer versions before eight) do not have the JSON object – baconjs should add a JSON object that performs the same tasks as the JSON object.

9.03 – `.querySelector` and `.querySelectorAll`

The functionality of the `.querySelector` and `.querySelectorAll` functions will be needed a lot in baconjs. As it is not possible to modify the HTMLElement prototype in Internet Explorer, the DOM function *`bacon_el`*`.select()` should replicate this functionality, and use `.querySelector` and `.querySelectorAll` wherever possible, as it is more efficient, more powerful, and faster.