



Behavioural Separation &



 **jQuery**

Progressive CSS

```
<p style="font-weight: bold; color: red;">This text is important</p>
```



Markup: `<p class="important">This text is important</p>`

CSS: `p.important { font-weight: bold; color: red; }`

```
<ul>
```

```
<li>Not important</li>
```

```
<li style="font-weight: bold; color: red;">Very Important</li>
```

```
</ul>
```



Markup:

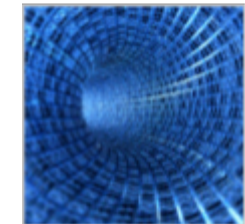
```
<ul>
```

```
<li>Not important</li>
```

```
<li><em>Very Important</em></li>
```

```
</ul>
```

CSS: `ul li em {font-style: normal; font-wieght: bold; color: red;}`



Progressive JavaScripting

- Do not make assumptions about dependencies
- Understand browsers and users
- Use Event Delegation
- Understand relationships
- Do not interfere with others
- Make maintenance easier

SOURCE: **Christian Heilmann**



Progressive JavaScripting - Why?

- Some phone browsers do not run JavaScript
- Some companies strip JavaScript at the firewall
- Some users use the firefox extention 'no script' for security.
- Rich JavaScript features may be inappropriate from many users with accessibility issues.
- Search engine spiders don't execute JavaScript, so if you (only) use JavaScript to load content into your page, they will never read or index your content.



Do not make assumptions about dependencies

- Don't expect JavaScript to be available.
- Don't expect browsers to support certain methods and have the correct properties. Test for them before you access them
- Don't expect the correct HTML to be at your disposal. Check for it and do nothing when it is not available
- Keep your functionality independent of input device
- Expect other scripts to try to interfere with your scripts. Keep the scope of your scripts as secure as possible.



Do not make assumptions about dependencies

Checking to make sure you can actually run the script in the user's browser by testing for method support before calling it:

```
if( document.getElementById ){  
scriptUsingGetElementById();  
}
```

```
if (document.getElementById('clickLink')!=null)  
document.getElementById...
```

Rather than adding this to every function the Filament Group have developed a script to test everything from the box model and various CSS properties to commonly used JavaScript Objects and AJAX capability. If these features are supported the relevant scripts are run and the enhanced stylesheet added and supported JavaScript functions included. Cookies are used to service repeat users with the correct enhancements quicker..



Do not make assumptions about dependencies

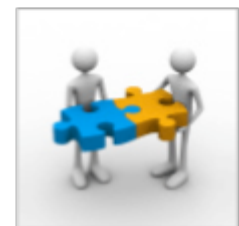
Use the DOM to create HTML on the fly in case you need it. An example of this are "print this" links - browsers don't offer a non-JavaScript way of printing a document, which is why you should create links like these with JavaScript.

The same applies to clickable headings that collapse and expand content. Headings can not be activated with a keyboard, but if you use JavaScript to inject links inside them even keyboard users can then collapse and expand the content sections.



Understand browsers and users

- Will my new interface work independent of input device, and if not, what should be the fallback?
- Is the new interface that I am building following rules of the browser or the richer interfaces it came from? (E.g. Can you navigate a multi level menu with your cursors or do you need to tab through it?)
- What is functionality that I need to offer but that is dependent on JavaScript?



Use Event Delegation

Event delegation and it has several benefits:

- You only need to test if a single element exists, not each of them
-
- You can dynamically add or remove new child elements without having to remove or add new handlers

You can react to the same event on different elements



Use Event Delegation

Event delegation (also) affects all the elements in the DOM hierarchy above the element you want to reach (this does not apply to all events though - focus and blur don't do that). This allows you to assign one single event handler to, for example, a navigation list - and use event handling methods to reach what element was really involved.

Another advantage of event delegation is that it avoids crippling the user's browser when you're working with a huge document. For example, if you have a table with thousands of cells, and you want something to happen when the user clicks on one, you won't want to attach a click handler to every single one of them. Instead, you can attach the click handler to a single table element and use `event.target` to pinpoint the cell that is being clicked.



Understand relationships

- How can I reach this element the easiest and with the least steps traversing the DOM?
- What element do I need to alter to reach as many child elements that I need to change?
- What attributes or information does a certain element have that I can use to link to another?



Do not interfere with others

Make sure your code does not have global function or variable names that other scripts can override.

The most basic is that you instantiate every variable using the var keyword.



Make maintenance easier

- Are all the variable and function names logical and easy to understand?
- Is the code logically structured? Can you "read" it from top to bottom?
- Are the dependencies obvious?
Have you commented areas that might be confusing?



Executing JavaScript

You cannot simply execute a function which is in an external file referenced in the head of the document if that script needs to effect events available in the DOM. It will run before the rest of the document has finished loading. The DOM methods won't work because there won't be any Document Object Model.

So you need to use the corresponding onload event handler, window.onload, to assign the function to this event:

```
window.onload = prepareList;
```

Rather than hogging the onload event for this one function, use something like Simon Willison's addLoadEvent function which allows you to queue up functions that you want to trigger when the document finishes loading.



Behavioural separation

- No in-line event handlers
- All code is contained in external .js files
- The site remains usable without JavaScript
- Existing links and forms are repurposed
- JavaScript dependent elements are dynamically added to the page



Behavioural separation

User agents that don't support JavaScript will choke on this href value:
`click here`

This still has a meaningless href value which doesn't function with JavaScript off:
`click here`

Now make the link meaningful even if JavaScript isn't enabled:
`click here`

Just as with CSS, you can add hooks to your markup to target the elements you want to play with:

```
<a href="arealpage.html" id="clickLink">click here</a>
```

In an external JavaScript file, you could now write a script to find the link with an ID of "clickLink" and have them execute a function when it is clicked.



Unobtrusive JavaScript



```
<ul id="list">
  <li><a href="images/image1.jpg">an image</a></li>
  <li><a href="images/image2.jpg">another image</a></li>
</ul>
```

```
document.getElementById("list").getElementsByTagName("a");
function prepareList(){
  if( document.getElementById '&&
    document.getElementsByTagName ){
    if( document.getElementById( 'list' ) ){
      var gallery = document.getElementById( 'list' );
      var links = gallery.getElementsByTagName( 'a' );
      for( var i=0; i < links.length; i++ ){
        links[i].onclick = function(){
          return showPic(this);
        };
      }
    }
  }
}
```



Unobtrusive JavaScript



Another example: Using a input's label as a 'hint text'

- Once the page has loaded, the JavaScript finds any label elements linked to a text field
- Moves their text in to the associated text field
- Removes them from the DOM
- Sets up the event handlers to remove the descriptive text when the field is focused



 *jQuery*



jQuery

\$

- \$(CSS Selector)
- \$(HTML)
- \$(DOM Element)



jQuery vars

store a value:

```
$("#div").click(function () {  
  var bgColor = $(this).css("background-color");  
  $("#result").html("That element is <span style='color:" + color +  
  ";>" + color + "</span>.");  
});
```

set values:

```
var n = 0;  
function () {  
  var cssObj = {  
    'background-color' : '#ddd',  
    'font-weight' : "",  
    'color' : 'rgb(0,40,244)'  
  }  
  $(this).css(cssObj);  
});
```

define a value:

```
var n = 0;
```



jQuery

Bob decides to paint his kitchen to match his car.



“I'll paint my kettle (what's the colour of my car?) and my toaster (what's the colour of my car?) and my dishwasher (what's the colour of my car?)”



“My cars blue. I'll paint my kettle, toaster & dishwasher blue”

```
var carColour = $('#car').css( "background-color" );  
function (paint-kitchen) {  
    $('#kettle').css({'background-color' : carColour});  
    $('#toaster').css({'background-color' : carColour});  
    $('#dishwasher').css({'background-color' : carColour});  
};
```



jQuery

Bob decides he's going to change his car's paint job.



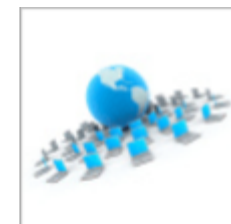
“The car's still blue, I can't do anything.”



“I know what the car's colour scheme will be - pink with a big yellow logo. I'll paint my kettle, toaster & dishwasher to match the car's colour scheme.”



```
function () {  
  var carColourScheme {  
    'background-color' : '#FF33FF',  
    'font-weight' : '900',  
    'color' : '#FFFF33'  
  }  
  $('#kitchen).css(carColourScheme);
```



jQuery

Some pronounce it vaar like jar - because you put suff into it which you retrieve later. It stands for variable because it's used for things that vary - in our example, the car will always have a colour scheme, but what that colour scheme is depends, usually, on how often Bob's wife crashes the car.



Computers can be forgetful. If you don't SET some information, all it takes is a semi-colon for the comuter to forget it.

If you GET some information, SET it in a var, then you can GET it again later when you SET it going with a function.



Unobtrusive jQuery

CSS & jQuery - .css for getting not setting

```
$("#div").click(function () {  
  var bgColor = $(this).css("background-color");  
  $("#result").html("That element is <span style='color:" +  
  color + ";>" + color + "</span>.");  
});  
  $(this).css({"border-style": "inset", cursor:"default"});  
  $(this).addClass("visited");  
✗ div#helpTips li.visited { border-style:inset; cursor:  
default; }
```



Unobtrusive jQuery

```
CSS: .animate{color:red;display:none};  
jQuery: $('animate').fadeIn();
```

This would use jQuery to fade in the elements that has the class name 'animate'. But if javascript is turned off, the CSS makes the browser hide the elements without ever showing them!

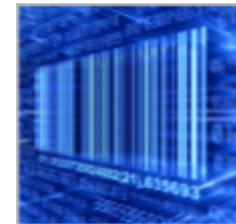
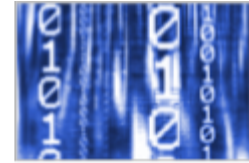
Instead, always make a habit of separating “javascript CSS” and “pure CSS”:

```
CSS:  
.animate {color:red}  
.animate_js {display:none};  
jQuery:  
$('animate').addClass('animate_js').fadeIn();
```



jQuery

- `jQuery(document).ready()` executes as soon as the DOM is ready
 - `$('#a.className')` uses a CSS selector to traverse the DOM
 - `.click(function() { ... })` deals with cross-browser event handling.
- It also avoids IE memory leaks.



jQuery selectors

Take advantage of jQuery's built-in custom selectors that are beyond basic CSS selectors:

:input example: get all the inputs on the page regardless if they are checkbox, textarea or select list - use :input

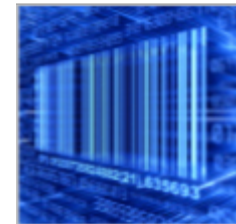
[attribute=value] example: find an input with the name, "container" - use input[name='container']

:eq(index) example: get the fourth table on the page - use table:eq(3)

```
$("#a").not("[href^=/]").not("[href^=#]").append("<span class='External'?</span>");
```

```
$("#a:not([@href*=mysite])").append("<span class='External'?</span>");
```

```
$("#h2 + p").addClass("Large");
```



Use Event Delegation

1. Events stop working after an AJAX request.

[An example of an event is a click handler on all links - `$('a').click(fn)`
]

2. Events don't work on a new, dynamically added element.

There are two solutions:

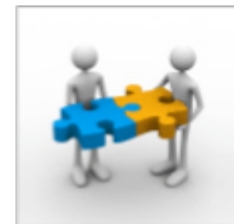
- a) Re-binding
- b) Event delegation



Use Event Delegation

```
$(document).ready(function() {  
    $('button.alert').click(function() {  
        alert('this is an alert message');  
    });  
});
```

Here we are registering the click handler for the button with a class of "alert" as soon as the DOM has loaded. So, the button is there, and we have a click function bound to it.



Use Event Delegation

But if we add a second `<button class="alert">` later on (with JavaScript) it will know nothing about that click handler. The click event had been dealt with before this second button existed so the second button will not generate an alert.



Two scenarios:

1. Events stop working after an AJAX request.

[An example of an event is a click handler on all links - `$('a').click(fn)`]



2. Events don't work on a new, dynamically added element.

To get the events to carry over to the new elements two common approaches are:

1. Event delegation
2. Re-binding event handlers.



Re-binding

```
function addItem () {  
    $('#list li.special button')  
    .unbind('click.namespacedAddition')  
    .bind('click.namespacedAddition', function() {  
        var $newLi = ('<li class="special"> <button>I am new</button> </li>');  
        $(this).parent().after($newLi);  
        addItem();  
    });  
}  
  
$(document).ready(function() {  
    addItemUnbind();  
});
```

If an element is added to the page (& the DOM) after the page is loaded event handlers need to be rebound manually. Event handlers are bound only once when the DOM is loaded. If you want additional links to behave as the original links, these 'loaded' elements must have all of their event handlers bound at the appropriate time. Re-binding creates a function that binds the handlers and then call it whenever new elements are introduced.



Re-binding

We call the function inside the click handler—and inside itself. That way, it will bind the event handlers to the new list item as well.



To avoid the multiple binding, we can unbind first and then re-bind. So instead of this ...

```
$('#list li.special button').click(function() {
```

... we'll have this ...

```
$('#list li.special button').unbind('click').bind('click', function() {
```



Note the use of `.bind()` here. This is the universal event binder that jQuery uses. All the others, such as `.click()`, `.blur()`, `.resize()`, and so on, are shorthand methods for their `.bind('event')` equivalent.

To stop the script unbinding the "non-rebinding" click handler as well, before it even had a chance to run once, apply a "namespace" to the click event for both binding and unbinding. So, instead of `.bind('click')` and `.unbind('click')`, we'll have, for example, `.bind('click.addit')` and `.unbind('click.addit')`.



Use Event Delegation

Getting Events to Embrace New Elements

```
$(document).ready(function() {  
  $('#list').click(function(event) {  
    var $newLi = $('<li class="special">special and new <button>I am  
new</button></li>');  
    var $tgt = $(event.target);  
    if ($tgt.is('button')) {  
      $tgt.parent().after($newLi);  
    }  
  });  
});
```

```
var $tgt = $(event.target);
```

- puts the target element in a jQuery wrapper and stores it in the \$tgt variable.



Use Event Delegation

Instead of attaching the `.click()` method to a button, we can attach it to the entire surrounding ``. Since the click event is registered with the unordered list (`#list`), it gets fired off when anything inside the list is clicked - including new elements added to the list after the page has loaded.

Through the magic of "bubbling," any click on the button is also a click on the button's surrounding list item and the list as a whole. It could be delegated to the containing div, and all the way up to the window object.

When we use event delegation, we need to pass in the "event" argument. So, in our case, instead of `.click()`, we'll have `.click(event)`.



Use Event Delegation

Event delegation can be used for binding event handlers after an AJAX load. We can bind the handler not to the elements that are loaded, but to a common ancestor element.

```
$(document).ready(function() {  
  $('body').click(function(event) {  
    if ($(event.target).is('h3')) {  
      $(event.target).toggleClass('highlighted');  
    }  
  });  
});
```

Here we bind the click event handler to the `<body>` element. Because this is not in the portion of the document that is changed when the AJAX call is made, the event handler never has to be re-bound. However, the event context is now wrong, so we compensate for this by checking what the event's target attribute is. If the target is of the right type, we perform our normal action; otherwise, we do nothing.



Use Event Delegation

Live Query

<http://plugins.jquery.com/project/livequery>

Live Query utilizes jQuery selectors by binding events or firing callbacks for matched elements auto-magically, even after the page has been loaded and the DOM updated.

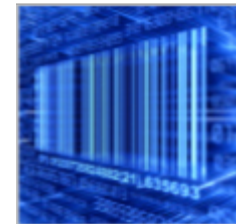
E.g. use the following code to bind a click event to all A tags, even any A tags you might add via AJAX. `$('a').livequery('click', function(event) { alert('clicked'); return false; });`

Once you add new A tags to your document, Live Query will bind the click event and there is nothing else that needs to be called or done. When an element no longer matches a selector the events Live Query bound to it are unbound.



jQuery & xml

```
$(function() {
  $('#update-target a').click(function() {
    $.ajax({
      type: "GET",
      url: "labels.xml",
      dataType: "xml",
      success: function(xml) {
        $(xml).find('label').each(function(){
          var id_text = $(this).attr('id')
          var name_text = $(this).find('name').text()
          $('<li></li>')
            .html(name_text + ' (' + id_text + ')')
            .appendTo('#update-target ol');
        }); //close each(
      }
    }); //close $.ajax(
  }); //close click(
}); //close $(
```



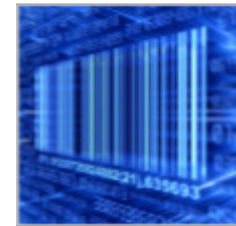
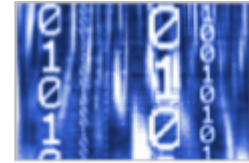
jQuery & xml

jQuery selectors return a collection of all matching elements, and the code takes the first (and in this case only) selected element and sets an `onClick` event handler. The event handler is given as a parameter to the `click` method, in this case an AJAX invocation to load the XML document using the special jQuery `$.ajax({})` construct.

The AJAX code is simplified with error handling has been left out.

The success function of the AJAX parameter structure starts when the document is successfully loaded.

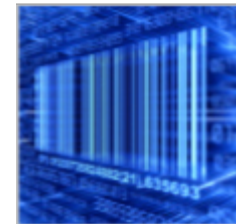
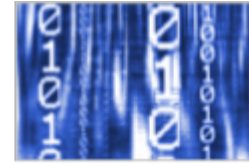
The resulting document is XML, and the function uses jQuery API to process that XML.



jQuery & xml

The find method applies a jQuery selector to a context, rather than the whole document, and in this case it's applied relative to the loaded XML, selecting all the label elements. The each method specifies an action to be performed over all the selected elements

For each label the id attribute's value is kept in a variable, using the attr method, and the text content of the name element is kept in another variable. The \$('') constructs an element on the fly and the html method adds HTML, or in this case plain text, as the element content. Finally, this newly created element is added to the target ol element.



jQuery tips

Compressing jQuery

Use the YUI compressor. Do not pack the code as the client-side decoding has significant overhead that outweighs the file-size benefits. Use JSLint to detect minor errors that can cause compressed JavaScript to fail.

Animations

When you do an animation on an element that animates the height or width (such as show, hide, slideUp, or slideDown) then the display style property will be set to 'block' for the duration of the animation. The display property will be reverted to its original value after the animation completes.

There are two common workarounds:

1. Use the fadeIn or fadeOut animations.
2. Add a float such that it appears to stay inline with the rest of the content around it.



jQuery tips

Advanced selectors / attributes

```
$("#a").not("[href^=/]").not("[href^=#]").append("<span  
class='External'>?</span>");
```

```
$("#a:not([@href*=mysite])").append("<span class='External'>?  
</span>");
```

```
$("#h2 + p").addClass("Large");
```



jQuery tips

The data method in jQuery allows you to associated data with an element on the page.

```
$('#selector').data('meaningfullname', 'this is the data I am storing');  
// then later getting the data with  
$('#selector').data('meaningfullname');
```

This allows you to store data with meaningful names and as much data as you want on any element on the page.

Use classes as flags.

If you aren't storing data, but need to set a flag on an element use a class. What do I mean by a flag? Well, for instance if you are in "edit mode" of a form you might use the class, "editing". With jQuery you can add a class with the addClass method and then check later if an element has the class with the.hasClass method.



 Fin.

