Module 3:

jQuery Methods and Events -- Case Studies

Some jQuery Lingo

implict iteration

Another way of saying automatic looping.

refactoring

Modifying existing code to perform the same task in a more efficient or elegant way. OK, this isn’t strictly jQuery, but it sure sounds impressive.

the *each()* function

To recap, jQuery supports automatic looping (or implicit iteration). For example, to make every <img> on a page fade out, only one line of code is needed:

```
$(‘img’).fadeOut();
```

But you will often want to loop through a selection of elements and perform a series of actions on each element. This calls for the **each()** function. For example, assume we have a sequence of 100 <a> tags similar to this:

```
<a href=”images/catPhotos.jpg” title=”Fluffy shredding the curtains”>
```

then jQuery could loop through all of these as follows:

```
$(‘a’).each( function() {
    var imageURL = $(this).attr(‘href’);
    var galleryImage = new Image();
    galleryImage.src = imageURL;
});
```

the *each()* function, cont.

For each <a> tag in the previous example, the following actions are performed:

1. return and save the contents of the ‘href’ attribute
2. create a new image object
3. load the image file (referenced by imageURL) into the image object

Note:

- **each()** takes an anonymous function as a parameter
- To access the current element through each loop, use the self-reference ***this***.

$(this) refers to a jQuery object within the object method’s body, in this case the current **<a>** element in the looping process. *The keyword **$(this)** will be used almost every time you use the **each()** function. Resistance is futile.*

In contrast, the keyword **this** will refer to a traditional DOM element if used within the each() function. Don’t confuse the two.*

*demo file available

Using events the jQuery way

In jQuery, most DOM events have an equivalent jQuery function. You probably recognize what these jQuery methods do:

```
$(‘a’).mouseover();
$(‘#menu’).click();
$(‘button’).dblclick();
$(‘#textfield’).focus();
```

...and they usually take an anonymous function as a parameter:

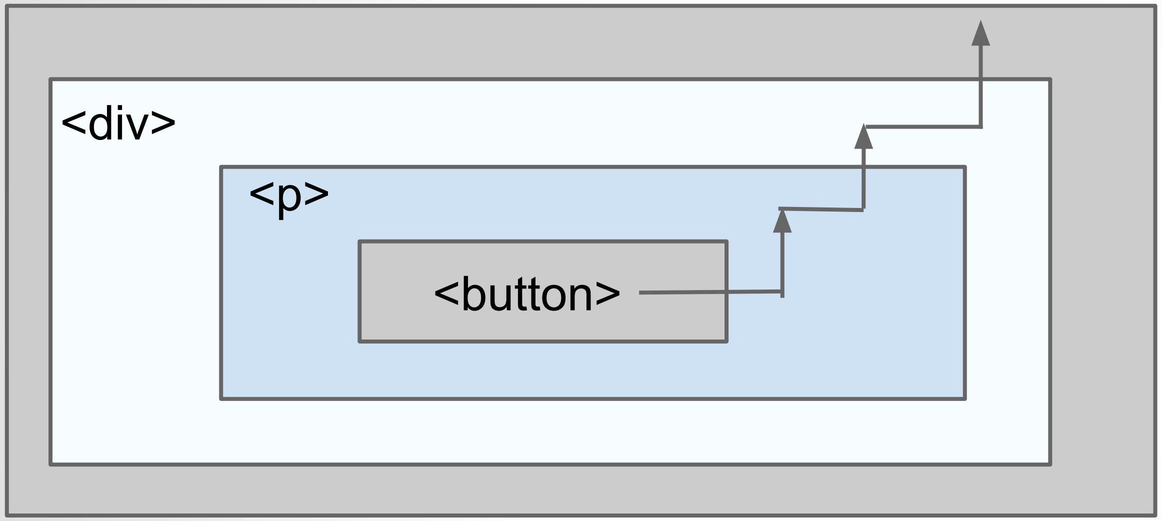
```
$(‘html’).dblclick( function() {
      alert(‘ouch!’);
});    // end double click
```

...or even 2:

```
$(‘#menu’).hover( function() {
      $(‘submenu’).show();
}, function() {
      $(‘submenu’).hide()
 });   // end hover
```

Event propagation

When an event occurs, a whole hiearchy of DOM elements gets a chance to handle the event. In jQuery, event capturing works as illustrated:



When the event occurs, it gets sent to the most specific element, and after this element has an opportunity to react, the event **bubbles up** to more general elements. In this example, if click event handlers were added to both the button and div (displaying an alert box in each case, say) and the button were clicked, alert boxes for both the button and div would be displayed!*

*demo file available

Event propagation, cont.

To prevent an event from passing onto ancestor tags in this case, jQuery provides the **stopPropagation()** function. It is a function of the event object (which contains information that was collected when the event occurred), so you would access it within an event-handling function:

```
$('#theLink').click( function (event) {
     // do something

     event.stopPropagation(); // stop the event from bubbling up
});
```

Advanced event management: the *on()* function

The **on()** and **off()** functions replace the **bind()** and **unbind()** functions which are now deprecated (so don't get in a bind over bind()).

In its simplest form, **on()** is equivalent to jQuery's event-specific functions like **click()** or **mouseover()**.

For example, these two are functionally the same:

```
$('#myButton').click( functionName );
$('#myButton').on('click', functionName);
```

on() provides more event-handling flexibility, since you can pass additional data for the event handler function to use.

Advanced event management: the *on()* function, cont.

The basic format of the on() function:

$('selector').on(eventName, myData, functionName);

Example Displaying an alert box in response to an event; the message in the alert box is different based on which element triggered the event.

```
var linkVar = { message: 'Hello from a link' };          // object literal
var pVar = { message: 'Hello from a paragraph' };         // another object literal
function showMessage( evt ) {
      alert( evt.data.message );                  // accessing the message property
};                                                // of the event data property

$('a').on( 'click', linkVar, showMessage );
$('p').on( 'mouseover', pVar, showMessage);
```

Note: Data passed to the **on()** function is either an *object literal* or a variable containing an object literal.

Other ways to use the *on()* function

1) Tie two or more events to the same function

Let’s say you have code that makes a large image appear when a visitor clicks a thumbnail image (the common “lightbox” effect; we’ll show you how to do this later in the course). Now you want the larger image to disappear when the visitor either clicks anywhere on the page or hits any key:

```
$(document).on(‘click keypress’, function() {
    $(‘#lightbox’).hide();
});  // end on
```

Other ways to use the *on()* function, cont

2) Attach several events that each trigger different actions

Yes, you don't need to use the on() function multiple times. For example, if you want to make one thing happen when a visitor clicks an element, and another when a visitor mouses over it, you could pass an object literal to the ***on()*** function as follows:

```
$('#theElement').on( {
	'click' : function() {
		// do something spectacular;
	},	// end click
	'mouseover' : function() {
		// do something breath-taking
	}	// end mouseover
}); // end on
```

Simulating user interaction

Sometimes it’s convenient to execute code that we have bound to an event, even if the normal circumstances of the event are not occurring. For example, suppose you had a div with an ID=”switcher” that you toggle to hide or show by clicking on a button:

```
$(“button”).click( function() {
	$(‘#switcher’).toggle( 1000 );
});
```

If you wanted the div to begin in its collapsed state, one way would be to simulate a click on the button using the ***trigger()*** function*:

```
	$(‘button’).trigger(‘click’);
```

NOTE: The click event handler on the selector object must already be defined when calling trigger()

*demo file available