AQUENT

GYMNASIUM

JQUERY BUILDING BLOCKS:
INTROSPECTIVE DYNAMISM

LESSON 5

ABOUT THIS DOCUMENT

This handout is an edited transcript of the *jQuery Building Blocks* lecture videos. There's nothing in this handout that isn't also in the videos, and vice versa. Some students work better with written material than by watching videos alone, so we're offering this handout to you as an optional, helpful resource.

Some elements of the instruction, like live coding, can't be recreated in a document like this one. We encourage you to use this handout alongside the videos, rather than as a replacement of them.

INTRODUCTION: INTROSPECTIVE DYNAMISM

Hello and welcome to an Aquent Gymnasium production of jQuery Building Blocks, 5 Ways to Cut Your Web Development Time in Half. I'm Dave Porter.

Lesson 5 is called Introspective Dynamism, which means changing one's self. Since JavaScript first appeared on the screen, web pages have had the ability to react to the user's actions. And jQuery provides a set of incredibly convenient event handlers to make it even easier to add dynamic features to your website. So another name for this lesson could have been Letting the User Do Stuff.

Now, adding dynamicness to your page requires listening for a lot of things to happen, and then doing a lot of different things in response. But today's overriding theme is going to be handling events. Key things that we'll be covering today include a deeper look at events; what they are; why we handle them, including clicks and drags; a look at the handy jQuery UI download builder, for when you want a little piece of jQuery UI, but not the whole thing; how to react as the user types; the jQuery clone method, which is a seriously convenient way to get a copy of any element, including its children; optionally, any event handlers and data that are attached.

Speaking of which, I will be covering jQuery's great data method, which allows us to attach arbitrary data to an element. For example, to point to another far away, but related, element. This takes advantage of a new HTML5 standard. And I'll also show you how to make sure that you're backwards compatible and cross-browser compliant here.

Today's table of contents;
Chapter 1, Atoms of a Click.
Chapter 2, Drag.
Chapter 3, … As You Type.
Chapter 4, Fresh Elements.

And we will conclude with the conclusion. Don't forget, I am not live. I am a video. So you can pause, go back, jump around as much as you need to. And please feel free.

Our last episode, as you may recall, ended with you driving off into the distance for a relaxing holiday weekend in the mountains while your friend Ethan, the office's designer and CSS guru, was stuck at the office implementing a feature.

This episode picks up one week later. It's Friday morning and you've arrived at the office. Right on key, your email chimes. It's from your boss, that guy with the bright ties and the angry forehead. He's attached a couple of screen shots, and the email says, "Ethan wrapped up his design for the poster customization process early.

Please turn your poster carousel browser into this by Monday. See spec."

The spec in question is for a new feature that everyone's very excited about. It's going to allow your company's customers to order posters with any caption, and as many captions as they want. The printing process is going to eat into margins a little bit, but everyone's confident that this is going to drive a major boost in revenues.

The spec itself specifies that your carousel items, when clicked, should pop up a pane that allows the customer to customize and position as many captions as they like. Ethan forwards you a solo response to the boss's email. "Happy Friday," it says. You look over at his desk. It's empty. He took the day off.

CHAPTER 1: ATOMS OF A CLICK

In Chapter 1, we are going to hook up the click event on the individual poster items, and to pop up the pane with the correct image, the one that was clicked on. We're also going to dig a little deeper into what makes a click.

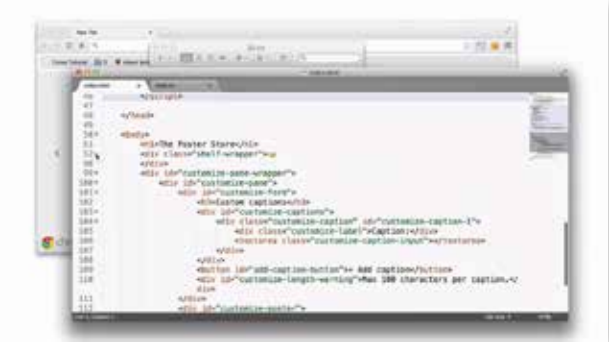
Now we've seen this command several times over the previous lessons. It's a real simple jQuery thing. It sets up a function to run every time the selected element is clicked. We used it to hook up our carousel's Forward and Back buttons before. And of course, we use it last time on Tabs.

The click method, by the way, it's just a convenience method, which means it's a simple sort of specific method that internally just makes a call to another, more complicated, method, in this case, the more generic on method, which sets up a handler for the named event. Well, we're going to be using this a little bit later. But for now, we're just going to stick to the convenience method.

We're also going to be using both the getter and setter versions of the jQuery CSS method. This allows us to see what the current value is for any CSS property, like display or background image, and to change it if we want. In general, we ought to keep this sort of thing into CSS classes, but today we're going to be doing it in a couple of places where classes are either overkill or impossible, because we're dealing with dynamic kind of things.

So let's dive right in. Please go to your lesson materials and open up the Index file and the CSS file in your IDE of choice. Now this is all the code from last time. We've got our hover effect all set up. We've got all the shelves, all that code, all that HTML is

in there. We don't care about it, so let's collapse it.



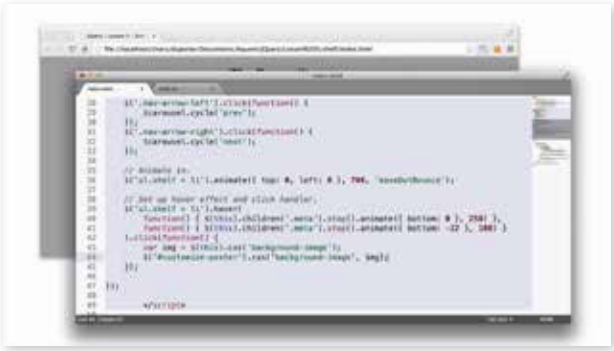
Then down to line 99, we pick back up with some new stuff, courtesy of Ethan. We've got our pane. It's inside our wrapper div, there's a click catcher. The click catcher's basically so the user can't interact with stuff in the background while the pane is open. And then inside of our pane, we've got a form side with some text area and a button. And we've got a poster side with a nice test caption waiting for us.

On the CSS side, here is all our carousel CSS again here. No changes. Notice that the display property of our pane's wrapper is currently set to None. This means that the HTML and the styles are all in place. But they're going to be completely invisible until we do something.

Note also, our caption text. It's set to white with a black shadow. But there's another class here for black with a white shadow. Spoiler alert. But keep an eye on that. And let's go ahead and pop that web page open in the browser, as well. Here it is. Here's our carousel. It works, just like last time. Perfect.

All right. We are going to now hook up the pane, so that it pops up when we click on any one of those flowers. Always remember to comment your code. We've already got our flowers selected here. So we can just chain that handler. We're going to add a click handler to these.

We're going to cheat a little bit here. We're going to extract the poster image directly from the CSS. Now in the real world, your images are going to be coming in from somewhere else and it's all going to be dynamic. We'll take a closer look at that next lesson. But for now, we're just going to extract them.



Note, by the way, here, note that we are doing our images with background image, rather than using an image tag. That's entirely up to you. That's a matter of preference. Using image tags makes it easier to save the image locally. It includes giving the user the ability to kind of drag the image out of the browser, which they do sort of intentionally or not. So in order to avoid that behavior, I prefer to use background image myself. But again, either way is fine. It's up to you.

And then finally, we are going to take that pane wrapper, and we're going to set its display CSS to block. Currently it's None. So this will show it. Now let's give it a test. We'll click on this guy. And it pops right up. Click the purple flower, you get the purple flower. So perfect.

Now we've got a spot here where the user enters the caption. We've got the poster on the right. There's a button we're going to use later. And note that the user can only enter the first 100 characters per caption, for space considerations or something.

At this point, I'd like to take a closer look at the click event that we've been throwing around with great abandon, just to give you a better understanding of what's going on, and to lay some foundational work for later on. A click, of course, is an event, meaning that the browser is letting your page know that something outside of the closed universe of your JavaScript code has happened. In this case, it's the user using the mouse.

But this chapter is called Atoms of a Click. And that's not just a great indie band name, because a click isn't the most atomic event. A click is a series of events. In this case, it's a mousedown, followed by a mouseup. And, by the way, without a mouseout event in the middle. Different places handle that differently. So if you want to do a double-click, right, it's just another two of these events strung together.

Events, again, are how the browser lets your page know that something sort of external has happened. We've looked closely at mouse button events that made up clicks. There's also events for cursor movement. There's keyboard events, you know, when the user hits the key or is typing. There's events for when the user interacts with forms. And there's even some events for non-user stuff, like network events. Those don't come from the user. And we'll cover those more next lesson.

So anytime you want to figure out how to interact with the user, just look deeper at what it is that the user is doing. Think through the events that would be involved. Most of them are actually really intuitive when you think about them.

CHAPTER 2: DRAG

So in Chapter 2, we're going to talk about drag, which is another thing the user can do with their mouse. If you want to let the user move something around, that's a drag. And a drag is made up of a mousedown followed by a whole little bunch of mousemoves. It's just a fire hose of individual events. And then a mouseup. Let's take a quick look at that.

So mousedown, a whole bunch of mousemoves, that's a whole bunch of individual events, and then a mouseup at the end there. And getting this to happen is our goal for this lesson.

Now it's important for you to get a good feel for the nature of what's going on here. But it turns out that doing drags from scratch is really tough. So let's see if we can find a plug-in that handles all the dirt for us. Now, as usual with UI stuff, jQuery has got something for us. Let's take a look.

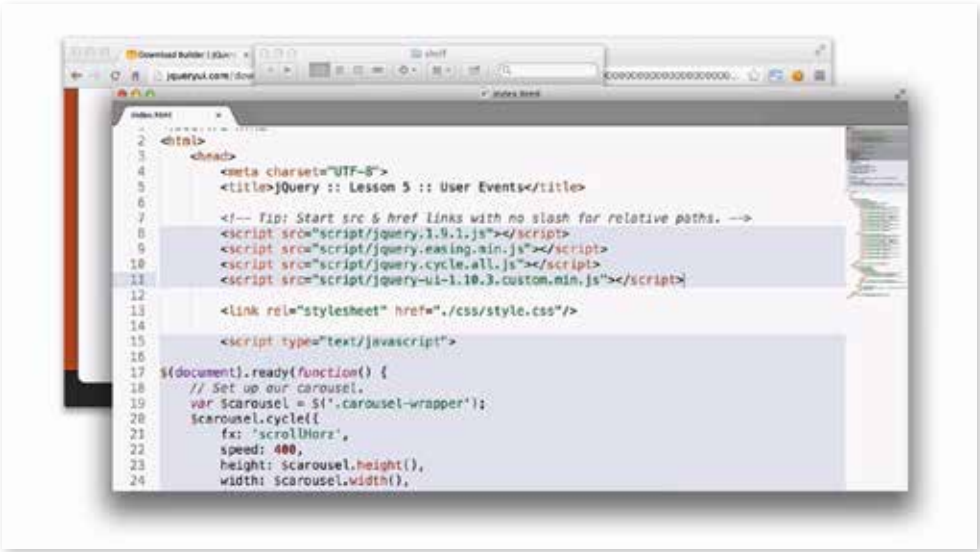
Now this guy here, draggable, it drags all around, including outside of the frame, which we don't want. We want to stop it at the edge of the poster. So let's take a look at constrain movement. If we take a look here, this guy drags around just within the box. That's perfect. That's exactly what we're looking for.

Great. So, unlike last time, jQuery UI is going to do exactly what we need it to this time. So let's take a

look at the custom download builder. We don't want the whole thing. We just want a minimal piece of jQuery UI. And jQuery UI's got this awesome download builder to let us do exactly that. In essence, going custom can ruin your cacheability. It's only worth it to do if you're saving a lot of bytes. But in this case, we just want the one feature and no theme. So it's going to be very worth it in this project.

If you later decide that you want to include the whole thing, don't forget to remove this file. Now note that the download includes a lot of extra helper files. And the only one we really want is to the minified script file itself. There it is.

So let's go ahead and copy that over into our project folder, like so. Drop there. Oops, make sure it doesn't do that. And then let's add it to our code. I'm just going to go back to this guy, pop him back open. You probably still have it open. And add a basic script tag here. Great. Now this is in our project.



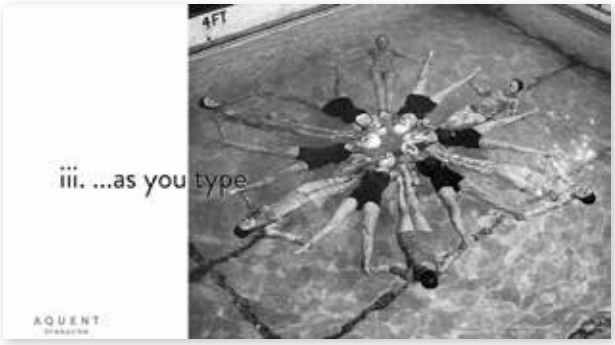
So with that in place, we now have access to the brand-new jQuery UI draggable method. Brand-new to us. That takes a hash of options. The only one we need right now is containment. We're going to specify that the element should be contained within its parent here. And there are other options for containment, which you can take a look at later.

So let's do it. And remember to always comment your code. I'm going to say that a lot, because it's important. And we are going to select all of the poster captions. For now, there's only one, but whatever. And we're just going to hit him with that draggable call. So let's do that. Give it a test. Pop open that same flower. And there we go. Perfect. It stays within its parent. Everything's great.

So by the way, like I said, that containment parent call is one of several different ways you can contain dragging. I linked the jQuery draggable docs from the reading material. So you can give that look.

And that is all it takes to add drag.

CHAPTER 3: ... AS YOU TYPE



Now in this chapter, we're going to synchronize what the user is typing into the text box with the actual caption on the poster. That means that this is also the chapter that we're going to be removing that test caption from the HTML. And we're going to let the user enter it themselves.

Now since we're looking to react to something that's happening from outside our code—in this case, the user is typing—that means that we're looking for an event. Now there's this change event that exists. And that sounds extremely intuitive. But unfortunately for us, that's an old event that sort of stuck around for compatibility. And it doesn't actually end up triggering until the user stops typing and leaves the box, which is not what we want at all.

So a quick trip to Google suggests that the event we want is propertychange. That's one word. So let's hook that up. Now, jQuery doesn't have a convenience function for propertychange like it does for click. So we're going to use our friend On. And let's add input to that as well. Google also suggests that this will give us compatibility with old versions of Internet Explorer.

Note, by the way, that just on the off chance this is crucial to your particular code, in IE9, both of these events will fire. In our case, we don't care. The code will run fine like that. But if your code can't handle that kind of double firing, you can just kind of include code in each event handler, which un-binds the other one when they're firing. Like I said, our code is going to work just fine here.

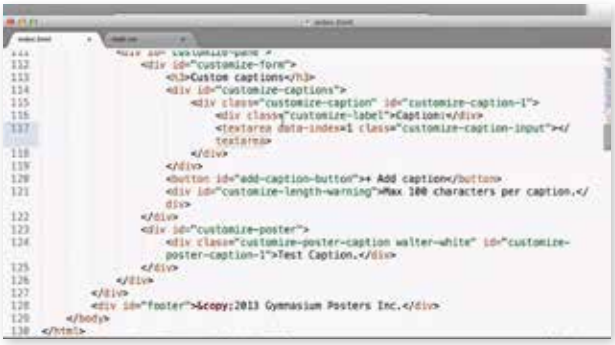
All right. Now we are still setting up the pane. Here's the On method to set up handlers for propertychange and input, for the folks that are stuck on Internet Explorer's earlier albums. Now we snag the value off of the raw element, just because we can. That is This, is the raw element. Then we have to wrap the element in This, in order to get at jQuery's convenient data function. And we're going to use that to grab the index out of the data attribute.

```
// Animate in.
$('ul.shelf > li').animate({ top: 0, left: 0 }, 700, 'easeOutBounce');

// Set up hover effect and click handler.
$('ul.shelf > li').hover(
    function() { $(this).children('.meta').stop().animate({ bottom: 0 }, 250) },
    function() { $(this).children('.meta').stop().animate({ bottom: -22 }, 100) }
).click(function() {
    var img = $(this).css('background-image');
    $('#customize-poster').css('background-image', img);
    $('#customize-pane-wrapper').css('display', 'block');
});
// Do initial setup on the customization pane.
$('.customize-poster-caption').draggable({ containment: 'parent' });
$('.customize-caption-input').on('propertychange input', function() {
    var val = this.value,
        index = $(this).data('index')
});
```

Now let's go down and we have to go ahead and add the data attribute to the initial HTML as well. And that's what this looks like. More on this in the reading materials if you're curious, by the way.

Than we do some quick validation on the label. Per our speck, we need to limit the value to 100 characters. And that is a super quick and easy way to do that. Always comment your code.



And finally, we snag the poster caption. We use the index to find it. And then we update a text like this. Great. Give it a check. Pop open that flower. And start typing. Wonderful. That looks good. All right.

Now that also means that we can go back into the HTML and remove that fake caption. That is now live, which is wonderful.

CHAPTER 5: FRESH ELEMENTS

Finally, some of our posters have got spots for more than one caption. So we'd like to support an arbitrary number of captions. Top and bottom, whatever. So on and so forth.

Now, if we knew that there would only ever be two or three, we could just show and hide them. But we want to support an arbitrary number. So we are going to learn how to take an existing element and create exact duplicates of them for use farther down.

Now, anytime you want to be able to support some arbitrary number of something, whether it's full-on form blocks like this, or like, say, posters in a carousel—see Lesson 6—then you are likely to want to use this sort of concept, or one of its many variations. We're also going to add the ability to close the pane in the background. And we'll get to that in a bit.

Now, there are many create-a-copy-on-demand processes that we can use. For simplicity's sake, we're going to use this one. It starts by taking an existing element and hitting it with jQuery's clone method, which just creates an exact duplicate. That first argument there saves us a few steps by specifying whether or not we also want to clone over our event handlers and our data attributes.

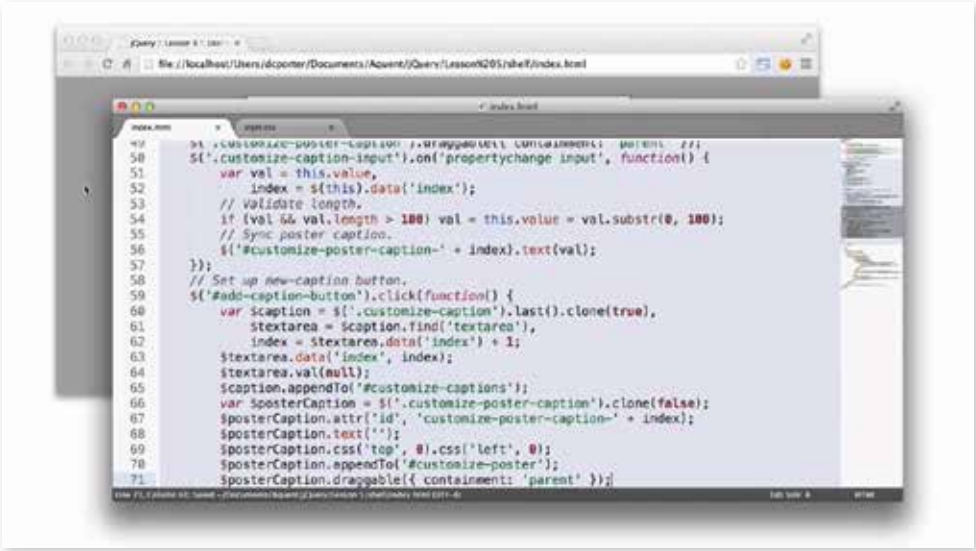
Now, usually, an exact duplicate isn't exactly what we want. So once we've got our clone, we're going to change some things. Then once our duplicate is all ready, we're going to append it. Now, there's two related append commands. The first one is simply called Append. And it's used to add new child elements to a selected element. Now the one we're going to be using is called Append 2. And it's basically the same, but in reverse. You use it to take an element and append it to a selected element. If you're having trouble remembering this, just sort of read the second one out loud. We want this element to append to its new parent.

All right. Let's do it. Now, we hook up the click handler on our lovely button element. Go ahead and grab that button. Now we're going to grab the last caption. And the reason to do the last one will become apparent in a second. And then we're going to clone it. And we're going to pass in a True, which means that we want the event handlers and the data, if there was any, to be copied over as well.

Next, we want to get the text area out of the caption that we've just cloned, and snag the index off of it, same as before. And we're going to increment it by giving it a little plus 1. Set the index to our new index. Now we're going to null out the closed caption. And take that caption and we're going up append it to the custom caption container.

Now we've got to do the caption over on the poster, too, so let's get it and then clone it. In this case, we don't want to clone the event handlers, too, because it messes with jQuery UI draggable.

Now we need to increment the captions ID, using that index that we already have, so the text area's event handler knows how to find it. Then we clear out its text. Let's reset its position at the top. And this is another OK use of CSS, because these are draggable values. So they are values that move along a continuum.



```
$('.customize-caption-input').on('propertychange input', function() {
    var val = this.value,
        index = $(this).data('index');
    // Validate length.
    if (val && val.length > 100) val = this.value = val.substr(0, 100);
    // Sync poster caption.
    $('#customize-poster-caption-' + index).text(val);
});
// Set up new-caption button.
$('#add-caption-button').click(function() {
    var $caption = $('.customize-caption').last().clone(true),
        $textarea = $caption.find('textarea'),
        index = $textarea.data('index') + 1;
    $textarea.data('index', index);
    $textarea.val(null);
    $caption.appendTo('#customize-captions');
    var $posterCaption = $('.customize-poster-caption').clone(false);
    $posterCaption.attr('id', 'customize-poster-caption-' + index);
    $posterCaption.text("");
    $posterCaption.css('top', 0).css('left', 0);
    $posterCaption.appendTo('#customize-poster');
    $posterCaption.draggable({ containment: 'parent' });
```

So now we append it to the poster itself. And let's go ahead and make it draggable as well. All right. Now we're going to give it a Save, give it a Reload, give it a look. Let's pop open the first guy, still. Let's add that caption. Let's drag it over to somewhere else. That looks good. And then let's go ahead and add a second caption. Look at that. That's awesome.

And we'll just go ahead and make sure that second caption is draggable as well. And it is. So we are looking good.

BONUS: CLICK TO CLOSE

OK. As a bonus, we're going to quickly add the ability to click in the background to close the pane. Ethan is probably going to notice the flaw in his design and add a button later on. But for now, this'll do. This will also give me the opportunity to show you something new and cool.

Obviously, we are going to be jumping back with our old friend, the click event. You've seen this guy a bunch, including earlier. But there's one more useful thing that you need to know about it that we're going to be using this time. Along with making this the element that's handling the event, jQuery also passes in an event object, which the browser generates anytime there's an event. Event objects are objects that come with a ton of information about the event, like coordinates of the click, the element that was originally targeted by the click. And we're going to need some of that information here in a minute. So don't touch that dial.

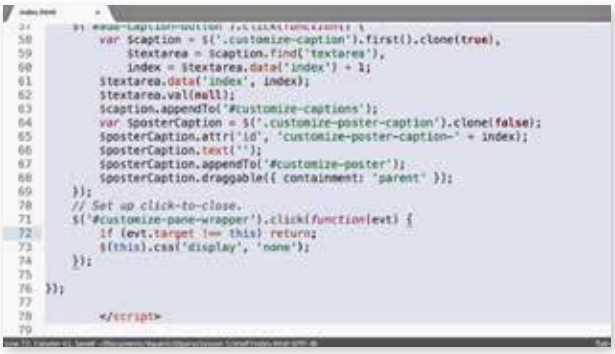
By the way, event objects are notorious areas of cross-browser incompatibility. So I'll tell you in a second how we're going to handle that here.

All right. Always comment your code. We are going to set up click to close. Let's first get that wrapper. And we're going to add the click handler to it. Should be really easy. Right? If we click in the background, make it go away. Let's give it a try. And we click in the background. And it disappears. That's great.

But something else happens. If we click in the foreground, it disappears then, too. And that's not what we're looking for. Now the reason that we're seeing this behavior is because, if you click on something and that element doesn't have an event handler, the browser is going to very happily ask that element's parent if it would like to handle the event. And so on up to the next level. And so on and so forth, all the way up until somebody has a handler for that event. This is called event bubbling, if you want to know the sort of abstract name for it.

So the solution to this is to ask the event object, which we're going to add, if the current element was the original target of the event. Click on the foreground and nothing happens. And that is exactly what we're looking for. By the way, I mentioned that event objects are notorious hives of cross-browser incompatability. So what are

we going to do to deal with that here? Here's our fix. We use jQuery, which provides its own standard event object. So this is one of those really wonderful moments where jQuery has made your life easier by papering over all these issues. And you don't even need to know about it.



And with that, we have successfully completed Ethan's Friday morning challenge. Good work.

So today, we built on top of our carousel, making a pane which appears with the correct image when it's clicked. We got the poster caption to auto-update as the user types. We let the user position it wherever they wanted. And we let them add enough extra captions to make for a very morose Shakespearean bee. And finally, we let the user close the pane by clicking in the background, without having that behavior from the foreground elements.

So in short, we took a page with a little bit of dynamic behavior, and we added a ton of dynamic behavior, all made super easy with jQuery.

Today we learned about a different way to set up click event handlers, which also extends to other events that don't have convenience functions. We learned about the CSS Getters and Setters, which you pass in the property name, like height or background color, to find out what the value is. Or you can pass in the property name and the value to change it.

We know we learned how to clone elements for reuse, including whether or not to copy over their event handlers. We learned more about events in general, which are how we get notified that the user or something else has done something that we need to pay attention to. We took a deeper look at click, which is made up of two events, and at drag, which is made up of three.

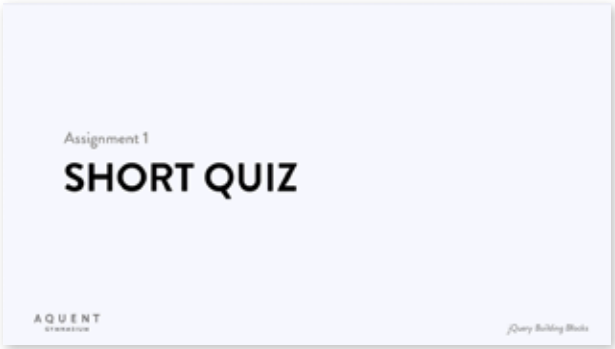
And we went and snagged jQuery UI's draggable behavior for our own nefarious purposes. And we learned how to contain it in the parent element. And we learned about the property change and input events, which together let you react to the user as she types.

And finally, we learned one more thing about events: that they can bubble up to their parents. So if you don't want an element's children to trigger events on the element itself, it's important to check the identity of the target element, to make sure that the user actually targeted the parent element itself.

We've got some assignments for you. First, please jump over to the lesson materials and take a short quiz. Secondly, when the pane goes away, we'd like to reset the form. Currently, if you close the form, open it back up, everything is exactly the way it was, except that the picture changed. So add code to reset the form whenever it's closed, or when it's opened, if you'd rather. Doesn't matter. Either way.

You want to remove all of the captions but one. And you want to clear that one remaining caption text. Don't forget to clear it in both places.

This should be a pretty quick one, although it will take a little cleverness to get the captions back down to one. And as usual, there are many ways to do that.



For your third assignment, please add some style. Our caption text is currently white with a black shadow. But even with the shadow, that doesn't look very good on all backgrounds, especially very light backgrounds. So for the third assignment, please give the user the ability to toggle this text from light to dark and back again.

Now here's a hint that I alluded to earlier in the lesson. I've already created a couple of CSS classes for you to use. You can find them in the CSS itself. So whether you use check-boxes or radio buttons, or some crazy thing of your invention, all the widgets should have to do is to swap classes. Don't forget to make sure that it still works when the user adds new captions. You should be able to set some different colors for each caption.

And by the way, switching between black and white text isn't the only style choice the user might want. If you're implementing this on your own page, you can use the same basic approach to add bold, italics, and, probably most importantly, in a bored-with-Times-New-Roman world, you can change out the font.

If you run into any issues, or if you just want to say hi, please do hop on the forum. Otherwise, we'll see you next time for Lesson 6, External Dynamism, when we will do similar internal changes, but rather than responding to user events, we're going to reach out to the vast Internet to get our updates.

This has been Lesson 5, Introspective Dynamism. I'm Dave Porter for Aquent Gymnasium.