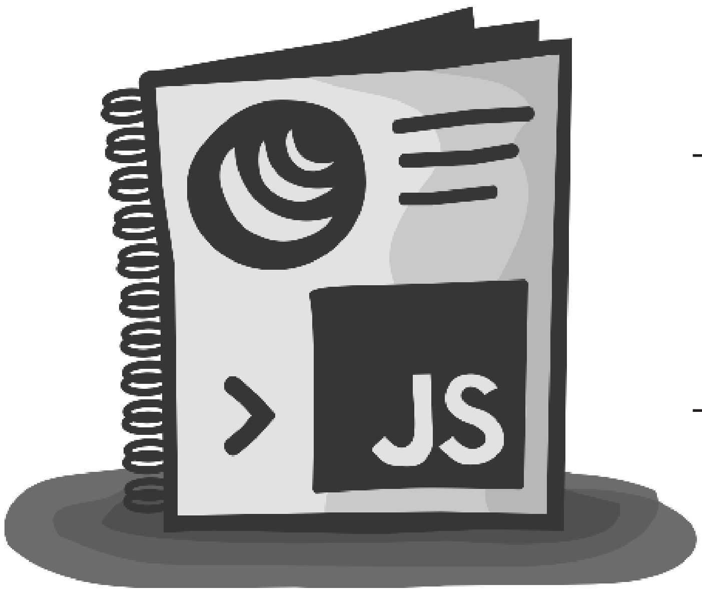


GYMNASIUM

THE JAVASCRIPT & JQUERY SURVIVAL GUIDE

Lesson 2 Handout

Managing Events

ABOUT THIS HANDOUT

This handout includes the following:

- A list of the core concepts covered in this lesson
- The assignment(s) for this lesson
- A list of readings and resources for this lesson including books, articles, and websites mentioned in the videos by the instructor, plus bonus readings and resources hand-picked by the instructor
- A transcript of the lecture videos for this lesson

CORE CONCEPTS

1. Events and event handlers can be thought of as tasks that you assign to various elements on a web page. For example when a user clicks on a list item, our event handler would execute one or more lines of code to do something (such as create an alert window.)

2. An anonymous function is code that you can pass to various events (such as the click event) and is in contrast to a named function.

3. The concept and usage of the word “this” in relationship to methods is an important one to understand. Typically the word “this” refers to the element that was clicked and more specifically allows you access to that element and in turn do something with it.

4. jQuery allows you to manually trigger an event using very simple syntax. Typically this syntax involves a method such as *click* or *double-click.*

5. You can remove a previously-attached event handler using the *unbind* method. One reason the unbind method is important is because it allows you to move event handlers that have already added to a DOM element.

6. The relatiionship between HTML forms and the server is important to understand. Specifially the idea that there is a round trip process involving submitting data to the server, then a delay of some sort and then a notification to the user of some sort that the data has been processed. Users have certain expectations about how these delays and notifications are handled and these expectations can be (partially) with JavaScript and/or jQuery code.

7. One technique that helps manage form behavior is the use of AJAX in order to provide useful feedback to users in real-time.

ASSIGNMENTS

- Build a simple AJAX-based form and add useful user feedback, such as providing error messages if a user doesn’t fill out certain fields, and an AJAX spinner to indicate that the form is processing the submission.
See pages 25–29 for detailed step-by-step instructions.

INTRODUCTION

(Note: This is an edited transcript of the The JavaScript & jQuery Survival Guide lecture videos. Some students work better with written material than by watching videos alone, so we're offering this 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.)

Lesson 2: Managing Events

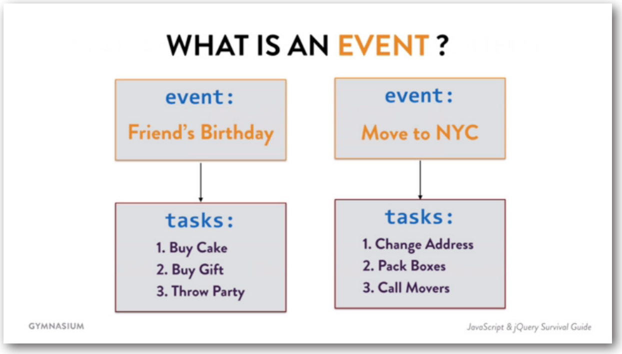
CHAPTER 1

EVENTS AND EVENT HANDLERS

GYMNASIUM
JavaScript & jQuery Survival Guide

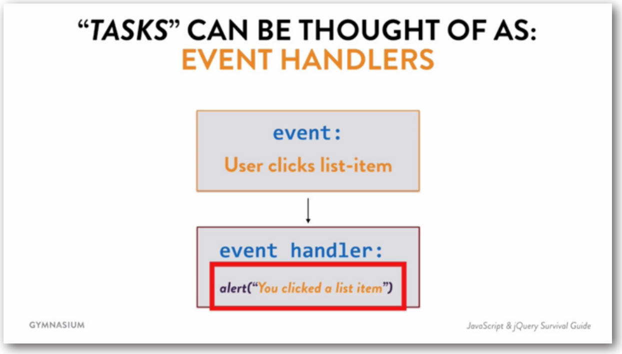
In this lesson, you'll begin to learn a number of techniques that you'll find useful on a day to day basis as a front end developer. Specifically, we're going to be looking at using jQuery to do things like manage event handlers. And at the end of the lesson, you'll be doing a deep dive into the use of jQuery and web forms.

At this point, I highly recommend that you follow along with each video using the code examples in your browser. Whenever I reference a code example, I'll be sure to let you know how to locate it in your local development environment. For this chapter, we'll be working with the file managing-events-with-jquery.js, which is in the Code Examples folder found in the root of the GitHub repository.



So let's take a moment to talk about events. What exactly is an event in a web page? Well, if we think about normal life, we have events that occur quite often. For example, whenever your friend has a birthday, you may want to buy them a cake, buy them a gift, and then throw them a party. In this case, you could think of your friend's birthday as an event and those actions as tasks that you associate with that event.

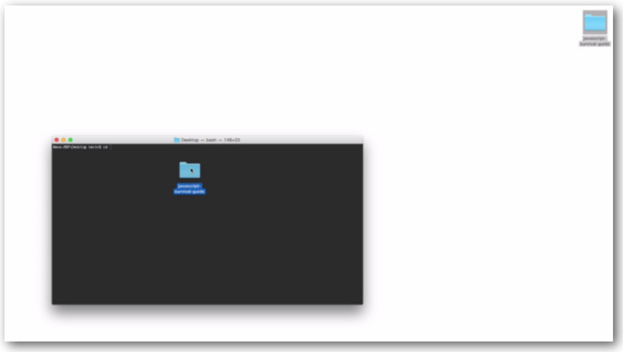
Another example would be moving to a new city. So the event would be the fact that you're moving, and then the tasks that you associate with that event would be changing your address, packing up all your boxes, and then calling the movers. So in both these cases, we have an event and then we have one or more tasks that we associate with that event.


"TASKS" CAN BE THOUGHT OF AS:
EVENT HANDLERS
event:
User clicks list-item
event handler:
alert("You clicked a list item")
GYMNASIUM
JavaScript & jQuery Survival Guide

Now, in web pages, we don't have friend's birthdays and we don't move to new cities, but we do have events such as a user clicking something. So in the case of, let's say, when a user clicks something, we could have a task that we associate with that event. We want to show an alert that says, you clicked a list item.

Well, in this case, we would have what's called an event handler that's attached to that event. So whenever the user clicks a list item, our event handler would execute one or more lines of code. And in this case, that code would be an alert that says, you clicked a list item. So it's really helpful to think of events and event handlers in terms of something that happens, and then a function that's executed whenever that thing happens.

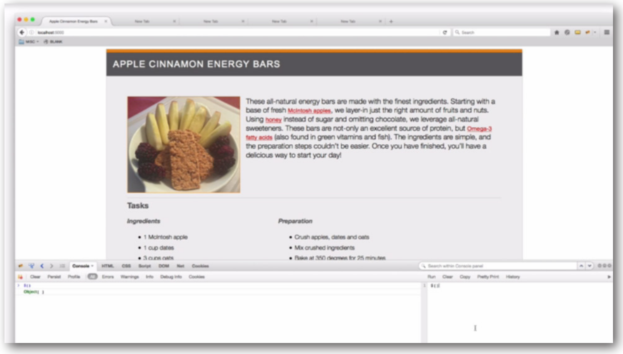
Now, I'm going to take a moment to start my local web server, just in case you've stopped it since the last video. And then I'm going to show you where to find the Code Examples file for the rest of this chapter. And I'll take a minute to kind of set things up so we can go back and forth real quick.



So first I'll open my terminal application. I'm going to type cd space and then drag my code repository folder to the terminal and hit the Enter key. And now I'm in that folder. And then I can simply type gulp serve, and that starts my local web server.

Now, I'm using Sublime Text Editor. You may use a different text editor, and that's fine. In my case, it's very easy for me to view all the files I want to work with if I drag my code repository folder to Sublime Text Edit. And I can open the Code Examples folder and double click managing-events-with-jquery. So I'm going to leave it there and it'll be easy to go back here when I need it.

And I'll open my browser and position it here, start my local web server by going to localhost:5000. And then I can kind of flip back and forth really easily. So we'll leave that there and I'll be working this way for the rest of this chapter.



APPLE CINNAMON ENERGY BARS

These all-natural energy bars are made with the finest ingredients. Starting with a base of fresh McIntosh apples, we layer-in just the right amount of fruits and nuts. Using honey instead of sugar and omitting chocolate, we leverage all-natural sweeteners. These bars are not-only an excellent source of protein, but Omega-3 fatty acids (also found in green vitamins and fish). The ingredients are simple, and the preparation steps couldn't be easier. Once you have finished, you'll have a delicious way to start your day!

Tasks

Ingredients

- 1 McIntosh apple
- 1 cup dates
- 3 cups oats

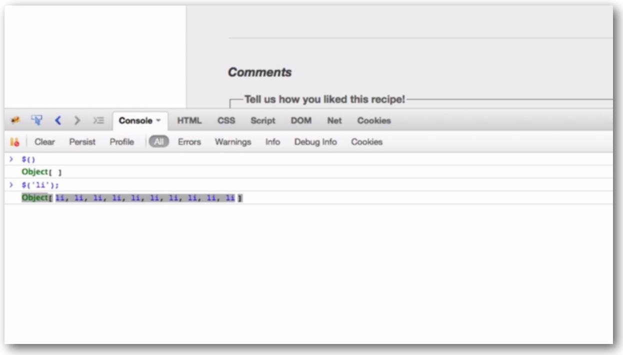
Preparation

- Crush apples, dates and oats
- Mix crushed ingredients
- Bake at 350 degrees for 25 minutes

```
$()
```

Open up your browser. And if you haven't done so already, navigate to localhost:5000 so we can see our example web page. And open the Firebug JavaScript console. Now, I'd like you to type $ open parentheses close parentheses. And that's it. And you can run this code.

Not too much happens. But what I want to point out here is the basic syntax that we're going to use to target dom elements in the page. In this case, nothing happened because we didn't say much to the jQuery function. We didn't pass it anything. We need to pass it something.



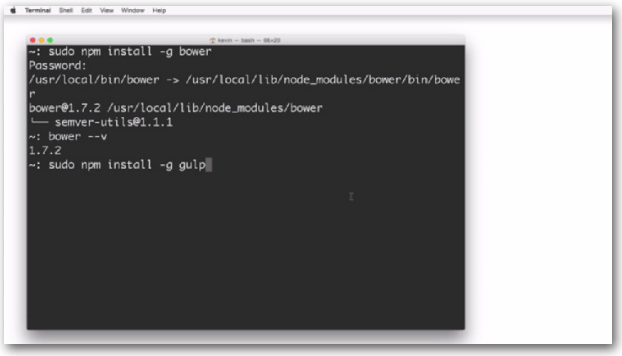
Comments

Tell us how you liked this recipe!

```
> $()
  Object[ ]
> $('li');
  Object[ li, li, li, li, li, li, li, li, li, li ]
```

So open your Code Examples file and copy the code example that's labeled jquery all list items. Go back to your web page and paste that in your JavaScript console. Now, when you run that code, a little more happens. We see that we get some stuff in the JavaScript console. That is the result of an expression.

The expression was basically we told jQuery, hey, target every list item in the page. And we passed it an LI, and it returns every list item in the page. Now, we could easily type the word body, and that would return the body tag. We can say, give me every div inside of the body tag, and we get some more stuff. And then we could say, give me every unordered list inside of any div that's inside of the body tag, and we see more stuff.



```
~: sudo npm install -g bower
Password:
/usr/local/bin/bower -> /usr/local/lib/node_modules/bower/bin/bowe
r
bower@1.7.2 /usr/local/lib/node_modules/bower
└── semver-utils@1.1.1
~: bower --v
1.7.2
~: sudo npm install -g gulp
```

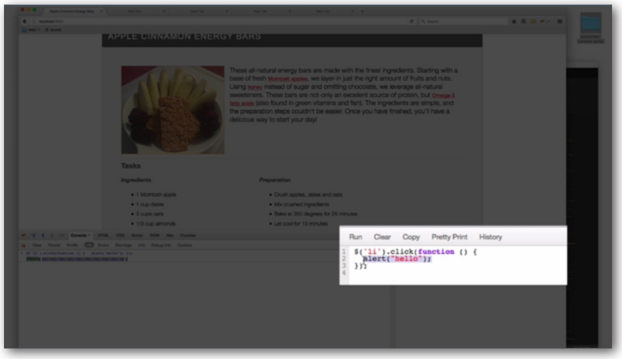
So if this syntax is starting to look familiar to you, it should, because as a front end web developer, you may be familiar with CSS. So the syntax that jQuery expects is virtually identical to CSS syntax. It allows you to use one or more selectors to be very specific in the elements that you target.

We're not going to be going too crazy on this in this course, but I just wanted to point out that the syntax is very similar to CSS. We can use classes. We can use IDs. And we can use nested hierarchies of elements in order to target the dom elements that we want in the page.

So refresh your page. Go back to your Code Examples file and select the code example labeled query all list items show click method. Copy that code. Go back to your JavaScript console. Paste it in and then run the code.

Now once again, not too much happened here, but what I want to point out is that the object that is returned from a jQuery query has one or more methods. In this case, we queried the click method and the JavaScript console showed us that that is a function. So we know that the objects that are returned from jQuery query have a click method.

Let's take it a step further. Go back to your Code Examples file, copy the code example labeled alert hello on click. And we can refresh the page. Paste that code in your JavaScript console and then run your code.



```
$('li').click(function () {
  alert("hello");
});
```

So once again, it returned all the list items in the page, because we're targeting all of the list items in the page. We're saying, hey, when any of those list items are clicked, I want to execute this function. This function has one line of code. It has an alert.

So what happens is, I think when we click any one of these list items, we should see an alert that says, hello. And that's exactly what happens. We click any list item and we see the word hello. Now, that's not too impressive, but the main thing I want to emphasize here is the syntax, because this is something we're going to be doing a lot in the remainder of this course.

And once again, the syntax is we target one or more elements with a query. We attach a method, in this case, the click method. And then we pass that method a function. And that function contains one or more lines of code. Think of those lines of code as tasks that we want to execute when something happens.

Now, we can also target, for example, an image. We know there's an image here. We know that's an img tag. So there's at least one img tag in the page. So we can say, you clicked the image. And then we can run our event handler. And now when we click the image, we see the alert.

We can target any element in the page that we want. The main thing to keep in mind is the syntax. Query the element, attach a method, and then pass a function to that method so that you can define one or more lines of JavaScript code that you expect to happen in your event handler. This function that you pass to the click method is your event handler.

In the next chapter, we're going to take a look at what an anonymous function is and how to use one. See you soon.

ANONYMOUS FUNCTION

Lesson 2: Managing Events

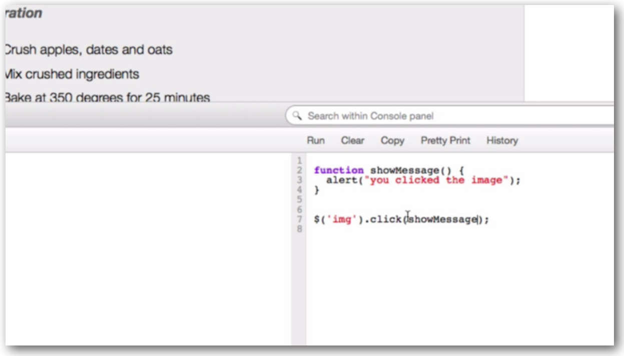
CHAPTER 2

ANONYMOUS FUNCTIONS

GYMNASIUM JavaScript & jQuery Survival Guide

So far in on our click event handlers, we've been using what's called an anonymous function. What exactly do I mean by anonymous? Well, the function that we're passing to the click method here has no name. And by name I mean, after the word function, we could type "show message." And that gives the function a name.

But in this case, it doesn't matter because the function is simply being passed right in and executed when the click method is executed. So it doesn't need a name. This name is kind of useless. It just has no real purpose or no added value.



ration

Crush apples, dates and oats

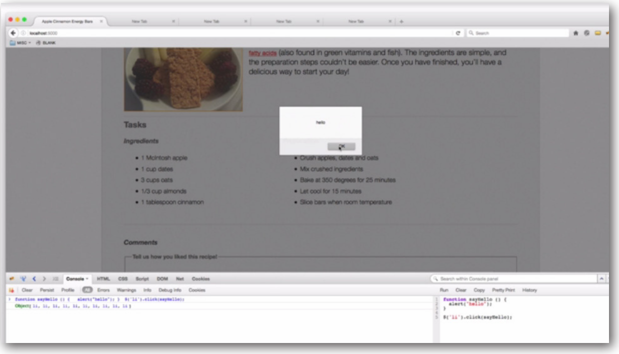
Mix crushed ingredients

Bake at 350 degrees for 25 minutes

```
function showMessage() {
  alert("you clicked the image");
}

$('img').click(showMessage);
```

If I were to take this function and move it outside of the click event method, give it a name, then pass that name to the click event method, then we've got an event handler. Instead of passing a function, we're passing the name of a function that exists already.

Let's check this out. Go to you code examples file. And copy the code example that's labeled Pass Named Function to click Callback. Paste that into your JavaScript console. Refresh the page. And I'll just mention, by the way, at this point, you can refresh the page by simply clicking the title. That might be a little bit easier for you.



Tasks

Ingredients

- 1 cup dates
- 3 cups oats
- 1/3 cup almonds
- 1 tablespoon cinnamon

- Mix crushed ingredients
- Bake at 350 degrees for 25 minutes
- Let cool for 15 minutes
- Slice bars when room temperature

Comments

Tell us how you liked this recipe!

```
> function sayHello () {   alert("hello"); } $('li').click(sayHello);
  Object[ li, li, li, li, li, li, li, li, li, li ]
```

```
function sayHello () {
  alert("hello");
}

$('li').click(sayHello);
```

So, once again, I've got a function called "Say Hello." It's being defined first. And then I've got my query to every last item. And I'm saying, for the click event, we're going to pass it this function, "Say Hello." So when I run this code, I click any list item. And I see the alert that happens.

So the end result is exactly the same. The only difference is, instead of passing a function to the click method, I'm passing the name of a function that we've defined already. Now, if I were to refresh the page and get rid of this name here and just take this function and paste it right in here and get rid of the name and then run this code, it's the exact same end result. I get my alert when I click any list item.

So we're going to be using this approach from now on, which is passing anonymous functions. But I just wanted to point out that you do have the option of passing an anonymous function to your click event method. Or you can pass what's called a named function expression, which will also work as well. It's just helpful to know that those options are available to you.

Next up, we'll continue to work with event handlers and look at how the simple word "this" contains an enormous amount of power in jQuery.

THE POWER OF 'THIS'

Lesson 2: Managing Events

CHAPTER 3

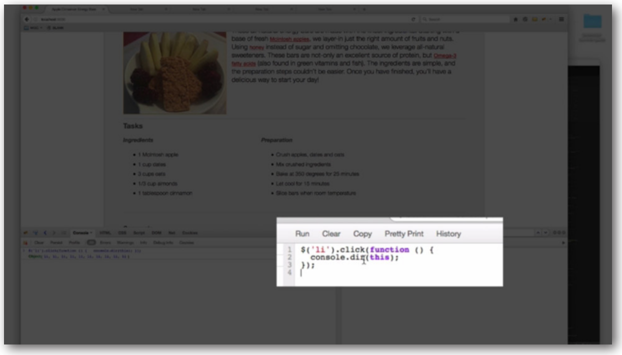
THE POWER OF 'THIS'

GYMNASIUM

JavaScript & jQuery Survival Guide

So now let's talk about how you learn about the item that was clicked or the item that's being iterated over in your event handler. Go to your code examples file and copy the code example labeled Inspect Clicked Element. So copy that code and then paste it in your JavaScript console and run it.

Now, what's happening here is we're using something called the console.dir method. And console.dir allows you to inspect an object. So it will show a message in the console, but it expects an object. And it allows you to look at the properties of that object.



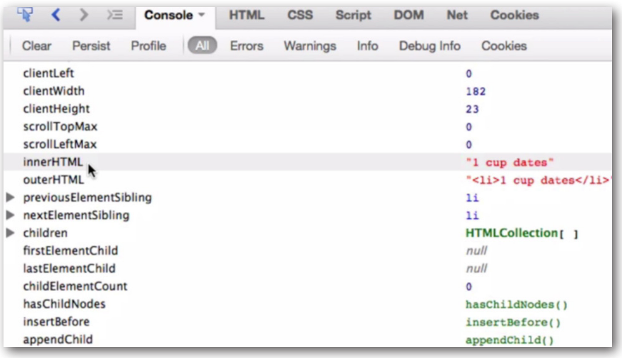
```
$('li').click(function () {
  console.dir(this);
});
```

We're passing the word "this" to that method. Now, the word "this" in JavaScript is very powerful. It has a different meaning at different times. It usually refers to the object to which a method belongs. Don't worry if that sounds kind of cryptic, but in this case, what it means is the word "this" refers to the element that was clicked. It's going to be an object and it's going to have some stuff. And we want to know about that stuff.

So when I click any one of the list items, I see a whole bunch of stuff in the JavaScript console, a lot of stuff. I clear my console and click another list item and I see more stuff. Now, most of this stuff doesn't really interest us. I mean, it's all interesting, but just for the sake of this conversation, what I'd like to know about is the text of the element.

So let me scroll down, and I'm going to look for a property called innerHTML. You'll have to scroll a bit, but will see eventually, there is the property, innerHTML. And look at the text, 1 Macintosh apple. So that does look like something I see on the web page.

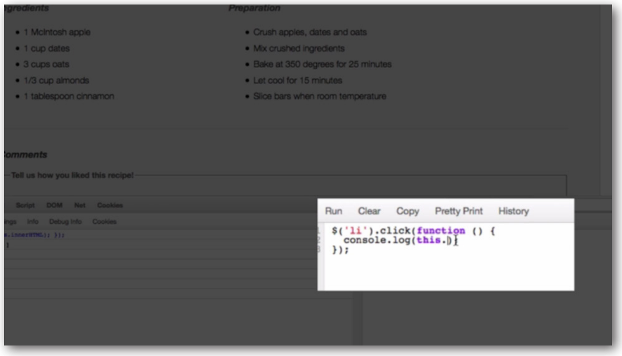
So let's click another item. I'm going to click 1 cup dates. So I scroll down and look for innerHTML. I'm going to see 1 cup dates. So innerHTML is a property of the object that was clicked. So when you click an element, this refers to the element that was clicked. And this is an object. And that object has many, many properties and methods. In this case, we're looking at one of them.



```
clientLeft                  0
clientWidth                 182
clientHeight                23
scrollTopMax                0
scrollLeftMax               0
innerHTML                   "1 cup dates"
outerHTML                   "<li>1 cup dates</li>
▶ previousElementSibling    li
▶ nextElementSibling        li
▶ children                  HTMLCollection[ ]
firstElementChild           null
lastElementChild            null
childElementCount           0
hasChildNodes               hasChildNodes()
insertBefore                insertBefore()
appendChild                 appendChild()
```

Let's go back to your code examples file and copy the code example labeled View InnerHTML of Clicked Element. So clear out your console, refresh the page, paste that code in the console, and execute it. So now when I click one of the list items, I see the actual text for that. Look at that list item. That's pretty cool.

So what we're using now is the console log statement. Console log is a method of console that just allows you to show a simple message in the console. In this case, I'm passing the innerHTML property of whatever element was clicked. And that's the word "this." So the word "this" provides access to whatever element was clicked. And then I'm going to pass it the innerHTML property. So I can see the text of any element that was clicked.

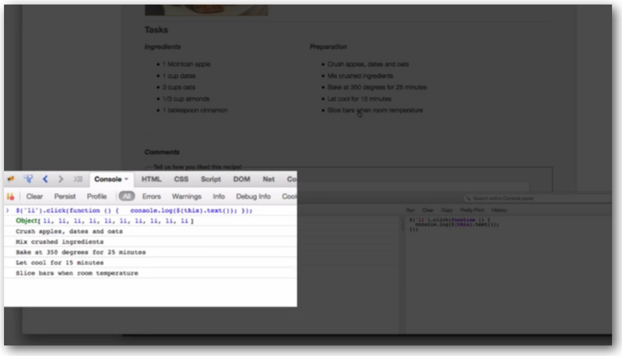


```
$('li').click(function () {
  console.log(this.)
});
```

This may seem like, why would I care about this? But I promise you, you will care. Because when you want to start doing more sophisticated scripting, JavaScript code in a web page, these kinds of methods are really going to become useful to you.

So go back to your example file and copy the code example labeled Execute Text Method on Clicked Element. Paste that in your JavaScript console and refresh the page. So now we're using the console log statement, and then we're passing it the word "this." But the word "this" is wrapped with dollar sign parentheses. So what's happening is we're executing the jQuery function again, passing it the word "this," which results in an object that has even more properties and methods than we would normally have.

And one of those methods is the text method. So when I execute this code and I click any one of the list items, I see the text for that list item. The end result is the same as using the innerHTML property, but it's a little more efficient. And the text method is definitely one that you'll use in the future when writing JavaScript code and using jQuery.



```
> $('li').click(function () {   console.log($(this).text()); });
  Object[ li, li, li, li, li, li, li, li, li, li ]
  Crush apples, dates and oats
  Mix crushed ingredients
  Bake at 350 degrees for 25 minutes
  Let cool for 15 minutes
  Slice bars when room temperature
```

But I wanted to point out the most important thing here is the power of the word "this." The word "this" provides access to the element that was clicked in your click event handler. And it will provide access to any element in any event handler when that code is executed. So it's really important to remember how powerful that word is. And it's something that we're going to see a lot in our upcoming chapters.

MANUALLY TRIGGERING AN EVENT

Lesson 2: Managing Events

CHAPTER 4

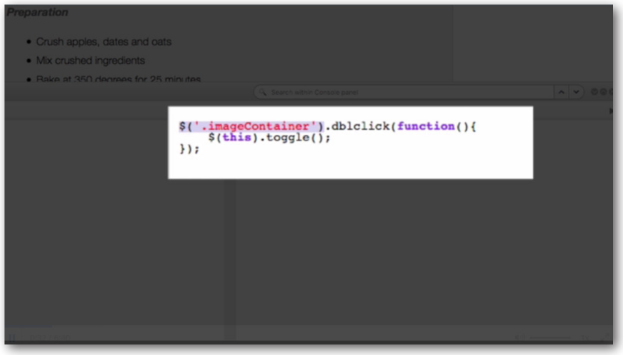
MANUALLY TRIGGERING AN EVENT

GYMNASIUM

JavaScript & jQuery Survival Guide

We've spent some time talking about event handlers. And that means that we have defined functions that will be executed when an event occurs. I'd like to switch gears a little bit and talk about how to manually trigger an event.

In your code examples file, copy the code example labeled toggle imageContainer and double click. In your browser paste that code example in. Now let's talk for a second about this.


Preparation

- Crush apples, dates and oats
- Mix crushed ingredients
- Bake at 350 degrees for 25 minutes

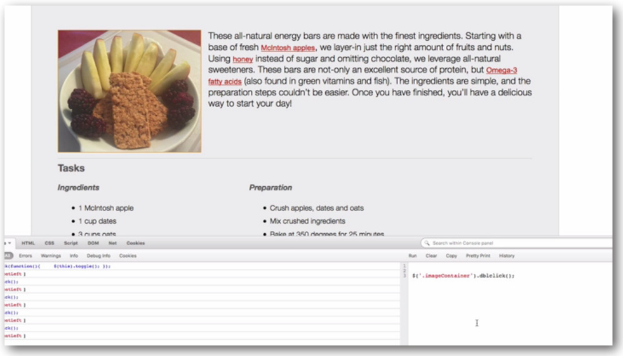
```
$('.imageContainer').dblclick(function(){
    $(this).toggle();
});
```

Here we're targeting an element with the class of imageContainer. Once again, the jQuery, the syntax for targeting an element is like CSS. So we're saying any element with the class of imagineContainer. In fact, if we inspect that element in the page, we can say here's the element itself that we're targeting. It's got the class of imageContainer.

So then we're chaining the DBL click method or the double click method. And what that's allowing us to do is to set up an event handler for the double click event. And I'm passing it an anonymous function.

What that means is when the user double clicks this image, or the element with the classic imageContainer, this function will execute it. And inside that function we're saying this—and once again, the key word this refers to the element that was clicked—toggle.

The toggle method is yet another jQuery method that is incredibly powerful. It allows us to change the visibility or display of an element in one line of code. So the toggle element says hey, if the element is visible, hide it. And if it's hidden, show it. So in one line of code we can do all that simply by using the toggle method.


These all-natural energy bars are made with the finest ingredients. Starting with a base of fresh McIntosh apples, we layer-in just the right amount of fruits and nuts. Using honey instead of sugar and omitting chocolate, we leverage all-natural sweeteners. These bars are not-only an excellent source of protein, but Omega-3 fatty acids (also found in green vitamins and fish). The ingredients are simple, and the preparation steps couldn't be easier. Once you have finished, you'll have a delicious way to start your day!

Tasks

Ingredients

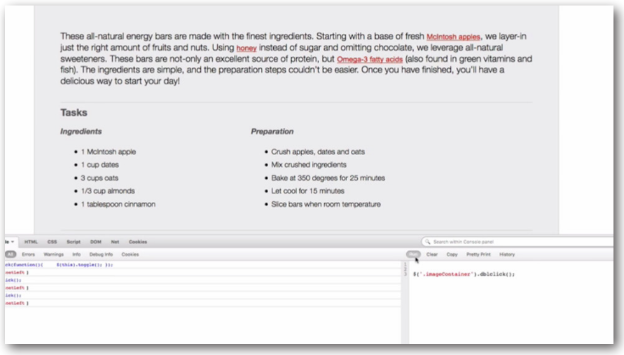
- 1 McIntosh apple
- 1 cup dates
- 3 cups oats

Preparation

- Crush apples, dates and oats
- Mix crushed ingredients
- Bake at 350 degrees for 25 minutes

HTML CSS Script DOM Net Cookies
Errors Warnings Info Debug Info Cookies
Run Clear Copy Pretty Print History
Search within Console panel

```
$('.imageContainer').dblclick();
```


These all-natural energy bars are made with the finest ingredients. Starting with a base of fresh McIntosh apples, we layer-in just the right amount of fruits and nuts. Using honey instead of sugar and omitting chocolate, we leverage all-natural sweeteners. These bars are not-only an excellent source of protein, but Omega-3 fatty acids (also found in green vitamins and fish). The ingredients are simple, and the preparation steps couldn't be easier. Once you have finished, you'll have a delicious way to start your day!

Tasks

Ingredients

- 1 McIntosh apple
- 1 cup dates
- 3 cups oats
- 1/3 cup almonds
- 1 tablespoon cinnamon

Preparation

- Crush apples, dates and oats
- Mix crushed ingredients
- Bake at 350 degrees for 25 minutes
- Let cool for 15 minutes
- Slice bars when room temperature

HTML CSS Script DOM Net Cookies
Errors Warnings Info Debug Info Cookies
Clear Copy Pretty Print History
Search within Console panel

```
$('.imageContainer').dblclick();
```

So when I execute this code, it's going to set up this double click event handler. And then when I double click the image, it should go away. Let's see what happens here.

I'm going to double click. I'm going to run the code first. The event handler has been set up. And I'm going to double click the image. And it went away. So that's exactly what we expected. So it's pretty cool.

The only problem is that now the image is gone. So we can't really enjoy this double click event handler anymore because we can't get the image back. We can't double click it again because now we can't see it. So I think we can fix that.

Back in your code examples file, copy the code example labeled manually trigger. Double click on imageContainer. Go back to your browser and paste that code in.

Now this is sort of similar in that we're targeting the exact same element—the element with the class of imageContainer. And we're chaining the DBL click or double click method. The difference is we're not passing it anything here. This section right here is empty.

So the difference is that when we call that event without passing anything, jQuery triggers or manually triggers that event. It's pretty cool stuff. We can use this same exact method to do two different things. In the previous example we chained the double click method and passed it a function. And here we're saying—we're just calling the method.

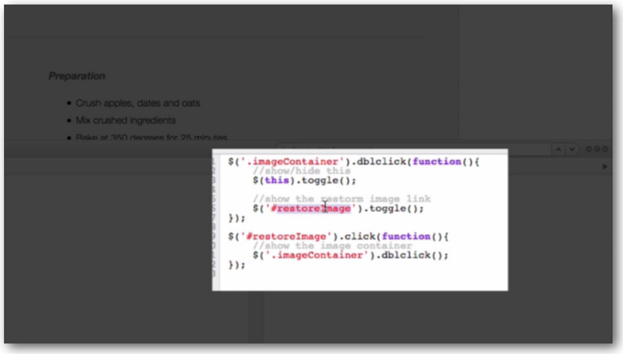
So if we do kind of like a side by side comparison, here up top is the previous example. And we're saying on double click, execute this function. And here we're just manually calling double click. So let's run this code. And now it brings the image back.

So this is pretty cool. We can double click the image, which makes it go away. And then we can run this event handler or manually trigger this event by calling the DBL click without passing it anything, which brings the image back. So we can kind of go back and forth, hide it, and then show it.

The only problem is this is not really practical. We're not going to ask the user to manually execute JavaScript code in the console. So I think there's a way to fix this.

Back in your code examples file, copy the code example labeled hide show image container top example. So paste this code in to your JavaScript console in your browser. And let's walk through this.

So we're setting up two event handlers here. The first event handler is a double click event handler. The second event handler is a click event handler. And the double click event handler—we're passing it an anonymous function.

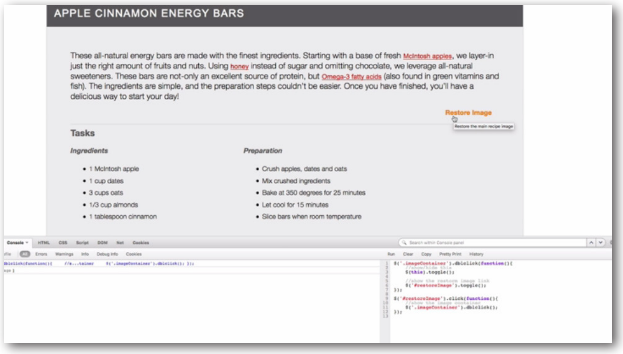

Preparation
Crush apples, dates and oats
Mix crushed ingredients
Bake at 350 degrees for 25 minutes

```
$('.imageContainer').dblclick(function(){
    //show/hide this
    $(this).toggle();

    //show the restorm image link
    $('#restoreImage').toggle();
});

$('#restoreImage').click(function(){
    //show the image container
    $('.imageContainer').dblclick();
});
```

And inside that function we're saying hey, when the user clicks this element, toggle the element. So if it's hidden, show it. If it's visible, hide it. And then the element with the ID of restore image—toggle it. So if that one's visible, hide it. If it's hidden, show it.

In the second event handler we're saying when the element with the ID of restore image is clicked, we're going to manually trigger the double click event on the element with the class of imageContainer. So from a visual perspective what I expect to happen here is that after I run this code, if I double click the image it should disappear.

And then I should see another element with the ID of restore image. And then when I click that element with the ID of restore image, it should manually trigger the double click event on the image, which runs this event handler, which hides—shows the image. And that's going to wind up hiding the element with the ID of restore image.



APPLE CINNAMON ENERGY BARS

These all-natural energy bars are made with the finest ingredients. Starting with a base of fresh McIntosh apples, we layer-in just the right amount of fruits and nuts. Using honey instead of sugar and omitting chocolate, we leverage all-natural sweeteners. These bars are not-only an excellent source of protein, but Omega-3 fatty acids (also found in green vitamins and fish). The ingredients are simple, and the preparation steps couldn't be easier. Once you have finished, you'll have a delicious way to start your day!

Restore Image

Tasks

Ingredients

- 1 McIntosh apple
- 1 cup dates
- 3 cups oats
- 1/3 cup almonds
- 1 tablespoon cinnamon

Preparation

- Crush apples, dates and oats
- Mix crushed ingredients
- Bake at 350 degrees for 25 minutes
- Let cool for 15 minutes
- Slice bars when room temperature

```
$('.imageContainer').dblclick(function(){
    //show/hide this
    $(this).toggle();

    //show the restore image link
    $('#restoreImage').toggle();
});

$('#restoreImage').click(function(){
    //show the image container
    $('.imageContainer').dblclick();
});
```

So let's run the code and see what happens. I double click the image. It went away as we expected. And then when I click Restore Image, the image comes back.

So I can go kind of over and over. Double click that, click this. Double click that, click this. And once again by double clicking this I'm executing this code. And by clicking this I'm executing this code.

So this is—I mean this is really, really helpful to be able to manually trigger an event. And you'd be surprised how often this kind of thing can—this kind of problem can come up.

A perfect example would be if we wanted to give the user a print version of the page and say print this page and allow them to hide the image when they print it because they may say look, I'm only interested in the recipe information. I don't necessarily need an image of the recipe in the page. And I want to save some ink.

So that would be one example of why you'd want to kind of do something like this. But the main thing is that the syntax for manually triggering an event with jQuery is incredibly simple. You simply call the event that you would use to set the event handler. By passing it nothing, you manually trigger the event.

Now in the next chapter we're going to look at the process of unbinding an event. I'll see you shortly.

THE UNBIND METHOD PART ONE: THE BASICS

Lesson 2: Managing Events

CHAPTER 5

THE UNBIND METHOD PART ONE: THE BASICS

GYMNASIUM

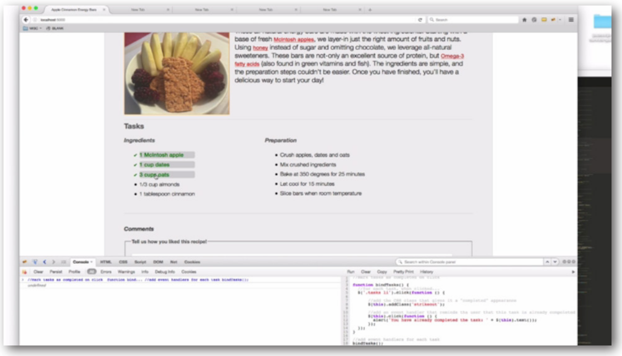
JavaScript & jQuery Survival Guide

We've spent quite a bit of time now, talking about how to create an event handler. We also discussed how to manually trigger an event. Now I'd like to cover how to unbind an event. And that's removing an event handler that we've already created.

In your code examples file, please copy the code example labeled—mark tasks as completed on click. Open your browser. Paste that code in. And then run the code. So now when I click one of the tasks, it looks quite different. It changes appearance.

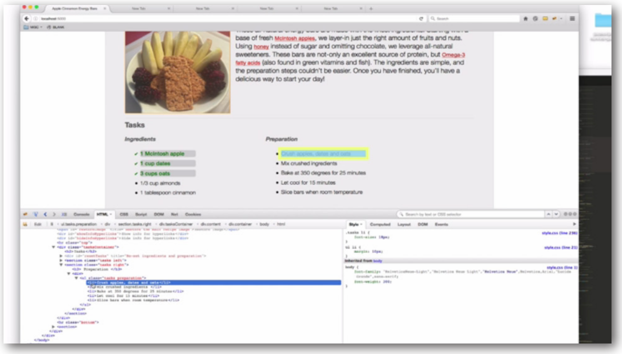
It looks like it's been completed. There's a great little check mark. And then it turns green with some shading behind it. So that's pretty cool. So let's look at how we did that. We have a function called bind tasks.

And we said—hey, any list item that's inside of an element with the class of tasks, we want to attach an event handler—click event handler. So we're kind of using a little bit more advanced targeting here, in our elements.



```
//mark tasks as completed on click

function bindTasks() {
  //for each task, when clicked...
  $('.tasks li').click(function () {

      //add the CSS class that gives it a "completed" appearance
      $(this).addClass('strikeout');

      //add an event handler that reminds the user that this task is already compoleted
      $(this).click(function () {
        alert('You have already completed the task: ' + $(this).text());
      });
  });
}

//add event handlers for each task
bindTasks();
```

So let's just kind of look at what this really means in the DOM. If you right mouse button click on any one of the list items, you can click Inspect Element with Firebug. It's one of the great features of Firebug is, you can inspect the DOM. And you can right mouse button click any element and go right to that element.

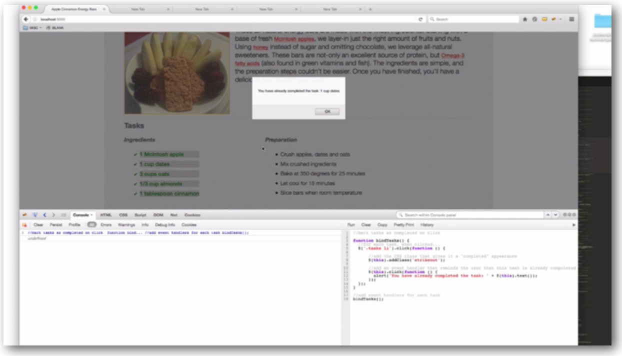


So when I click this, I go right to that list item. And as I can see, these list items are inside of an unordered list with the class of task. So we're targeting any list item that's inside of an element with the class of tasks. Got it. OK, cool.

So when any of those elements are clicked, we're going to execute some code. The first thing we're going to do is—we're going to reference the element that was clicked with the this keyword. And we're going to call a new method called the addClass method.

The jQuery addClass method allows you to add a CSS class to an element that, in this case, was clicked. So if I click one of the elements that was already clicked—right mouse button click and then choose Inspect Element with Firebug.

I can see it's got this class strikeout. So we added the class strikeout. And watch what happens when I click one third cup almonds. That's going to happen right down here. Watch what happens. It gets a class strikeout. And if I click the next one—one tablespoon cinnamon—it doesn't have a class strikeout. I click it.



```
//mark tasks as completed on click

function bindTasks() {
  //for each task, when clicked...
  $('.tasks li').click(function () {

      //add the CSS class that gives it a "completed" appearance
      $(this).addClass('strikeout');

      //add an event handler that reminds the user that this task is already compoleted
      $(this).click(function () {
        alert('You have already completed the task: ' + $(this).text());
      });
  });
}

//add event handlers for each task
bindTasks();
```

Now it's got a class strikeout. So the jQuery addClass method allows you to add a CSS class to a DOM element. Next, we create an event handler on the fly.

So in our event handler for the click we said—hey, the element that was clicked using the this keyword—when that element is clicked, we're going to create another event handler that's going to be an alert that says—you have already completed the task.

And then once again, we use the word—the this keyword and a text method. So it's kind of a custom element. So I think that means that when we click any of the elements that was already clicked or marked as completed, we're going to get a message that says you have already completed the task.

And then that element's text. So I think if I click one Macintosh apple again, I should see an alert that says, you have already completed the task one Macintosh apple. So let me do that. And that's exactly what I see.

So great, that worked.

And if I click the element one cup dates, I should see an alert that says, you have already completed the task one cup dates. And that's exactly what I see. But when I click one of the uncompleted tasks, the first thing that should happen is that it changes appearance because it hasn't been clicked yet.

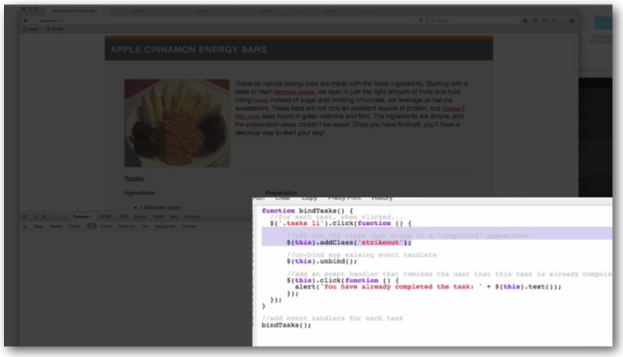
So that's going to happen here. So if I click crushed apples, dates, and oats—first, it changes appearance. And then a click event handler was added. So when I click it again. I get the reminder—you have already completed the task crush apples, dates, and oats.

That's great. And when I click it again, I get the alert again. Or—again. And then when I click it again, I get it three times. So wait a minute, every time I click this element I'm getting the alert. But I'm getting it one extra time every time I click it.

That's not good. That's not we want. We only want to see the alert once. But now that I've clicked it five times, I'm seeing the alert five times. I click it a sixth time. I'm seeing the alert six times. Something's not right. That's a bug. And what's happening is—this event handler is getting added every time we click the event.

So every time—every time I click the elements. Every time we click one the tasks, a new click handler is being added. So we don't want that. We don't want multiple alerts to happen. We just want one alert. So let's fix this bug.

Go to your code examples file. And click—I'm sorry—copy the code that's labeled—mark tasks as completed on click with unbind. So copy that code. Refresh the page. Paste that code in. And let's see what's changed here.



```
function bindTasks() {
  //for each task, when clicked...
  $('.tasks li').click(function () {

      //add the CSS class that gives it a "completed" appearance
      $(this).addClass('strikeout');

      //un-bind any exising event handlers
      $(this).unbind();

      //add an event handler that reminds tha user that this task is already compole
      $(this).click(function () {
        alert('You have already completed the task: ' + $(this).text());
      });
  });
}

//add event handlers for each task
bindTasks();
```

What's changed is that—on the click event handler for the list item, after adding the class strikeout, we say this unbind. So once again, this—the element that was clicked—and then it's a new method called unbind. And the unbind method is a jQuery method that says—remove any existing event handlers for this element.

So at this point, there's no more event handlers for the list item. And then we add our event handler that says—hey, you already clicked this item. You've already completed this task. So I think now we should have the behavior that we want.

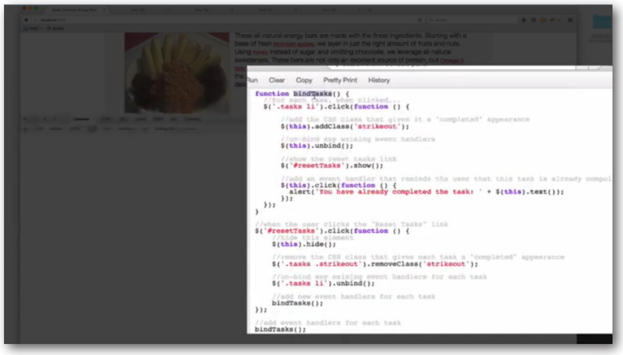
So I'm going to click an element. And it's marked as completed. Click it again. And I get the reminder that I've already completed this task. And I click it again and I get the reminder. But I only get the reminder once. And that's exactly what we want.

On the first click, change appearance. Second click, I get through a minder, but I only get the reminder once. And that's because we used the unbind method to remove the previous event handlers. So with every click, we're removing the previous event handlers and adding a new event handler.

So that's pretty cool. We fixed the bug. But let's say we want kind of like a real world scenario. Why would we do this? What situation would come up where we'd would want to unbind an event?

Well, let's imagine that the user wants to be able to mark off things as completed—I've already completed this item, but now I'm kind of halfway through the recipe, and I want to start all over. And I want to kind of go back. And I haven't completed any tasks.

So we want to give the ability to kind of reset—start from scratch. So refresh the page. Go to your code examples file. And copy the code example labeled—add reset tasks link. Copy that code. Go back in the page. And paste that code in your JavaScript console.



```
function bindTasks() {
  //for each task, when clicked...
  $('.tasks li').click(function () {

      //add the CSS class that gives it a 'completed' appearance
      $(this).addClass('strikeout');

      //un-bind any exising event handlers
      $(this).unbind();

      //show the reset tasks link
      $('#resetTasks').show();

      //add an event handler that reminds the user that this task is already compl
      $(this).click(function () {
        alert('You have already completed the task: ' + $(this).text());
      });
  });
}

//when the user clicks the "Reset Tasks" link
$('#resetTasks').click(function () {
    //hide this element
    $(this).hide();

    //remove the CSS class that gives each task a 'completed' appearance
    $('.tasks .strikeout').removeClass('strikeout');

    //un-bind any exising event handlers for each task
    $('.tasks li').unbind();

    //add new event handlers for each task
    bindTasks();
});

//add event handlers for each task
bindTasks();
```

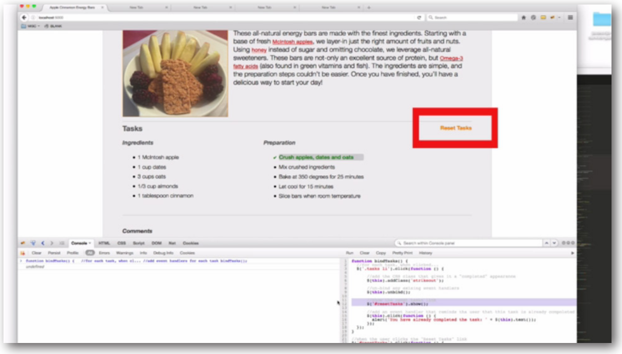
And let's take a look at the code before we do anything. So we've got this function, bind task. A lot of it looks familiar. Any list item inside the element with the class tasks when it's clicked, we're going to add the strikeout class, which gives it that appearance that it's changed.

Then we're going to unbind any previous event handlers. And then we're going to show some element called reset tasks. Now, reset tasks—that's a hash, I see here. A little different than the dot. So dots indicate classes. Hashes indicate IDs. Once again, the syntax for targeting elements is virtually the same as CSS.

So this is targeting an element with the ID of reset tasks. OK. And we're using a new method called show. And that's saying—show this element. If it's hidden, show it. And I'm assuming it's hidden because I don't see any element that looks like it's reset task.

So that should be something new that we should see. And then we're going alert—hey, you've already completed this task. Great. But now I've got a new event handler. So this element with the ID of reset tasks—we're creating a click event handler for this element.

And we say—when this element is clicked, hide it. And then we're going to look for any element that has the class strikeout that's inside an element with the class tasks. And we're going to remove that class strikeouts. So remove class is very similar to add class.



These all-natural energy bars are made with the finest ingredients. Starting with a base of fresh McIntosh apples, we layer-in just the right amount of fruits and nuts. Using honey instead of sugar and omitting chocolate, we leverage all-natural sweeteners. These bars are not-only an excellent source of protein, but Omega-3 fatty acids (also found in green vitamins and fish). The ingredients are simple, and the preparation steps couldn't be easier. Once you have finished, you'll have a delicious way to start your day!

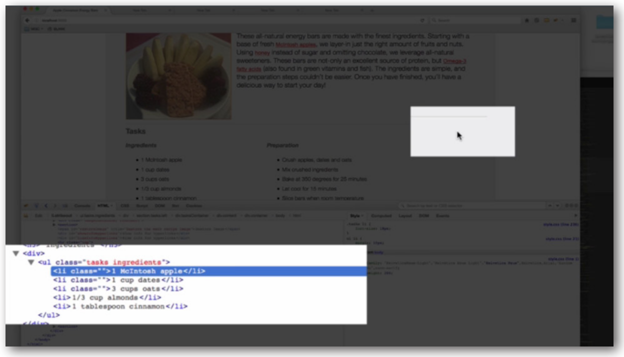
- 1 McIntosh apple
- 1 cup dates
- 3 cups oats
- 1/3 cup almonds
- 1 tablespoon cinnamon

- ✓ Crush apples, dates and oats
- Mix crushed ingredients
- Bake at 350 degrees for 25 minutes
- Let cool for 15 minutes
- Slice bars when room temperature

But it says—hey, if an element has a class, or that class and we pass it the name of the class we who want to remove. And then we unbind all of the event handlers for the list elements. And then we bind the tasks again. So that's a lot of stuff. Let's see it in action and we'll kind of just verify everything we expected.

So I'm going to run the code. I'm going to click an element. And the second I click the first element, I see this reset tasks link. And that happened because we said—reset tasks, show. The element that has the ID reset tasks—show it.

So we see that element. Cool. And once again, if I click any one of these tasks, they are marked as completed. And if I click any completed tasks a second time, I get the alert saying—hey, you've already completed the task three cup oats or completed the task one cup dates. Great.



```
▼ <div>
  ▼ <ul class="tasks ingredients">
      <li class="">1 McIntosh apple</li>
      <li class="">1 cup dates</li>
      <li class="">3 cups oats</li>
      <li>1/3 cup almonds</li>
      <li>1 tablespoon cinnamon</li>
    </ul>
```

But when I click Reset Tasks, look what happens—everything goes back to normal. All the tasks that were marked as completed kind of lose that appearance and kind of go back to normal. And the Reset Tasks link disappeared as well.

So let's take a look at that. So let's click a couple of tasks. And these tasks, when they were clicked, it said—add class strikeouts. Let's just kind of confirm that. So if you click one of the completed tasks with your right mouse button. Choose Inspect Element with Firebug.

You'll see they all have the class strikeout. OK, cool. So that's what we expect. The code add class worked on those elements. Now watch what happens when I click Reset Task. I'm going to click this link here. And if you look at these three elements—they all have the class strikeout—watch what happens when I click Reset Tasks.

They all lose the class strikeout. And by losing the class, all that CSS that gave them that completed look with the check mark and the green text, went away. So the remove class method is pretty cool. And it says—if an element has a class, remove it.

And then we unbound all of the list elements—list items. And then we rebound them again. So that gave us the ability to get the exact behavior that we want. We wanted to give the user the ability to say—I've already added these ingredients.

I've already completed these tasks. If I accidentally click a task I say—oh, that's right. I already completed that task. Great. So I've got a second click handler for all the completed tasks. And then if I say—hey, I want to start from scratch, I can just click Reset Task and start over.

So this functionality that we've put together in this example, it relies heavily on a few methods. We've already talked quite a bit about the click method, but the addClass method is definitely one that's important here. The unbind method is super important because it allows us to move event handlers that we've already added to a DOM element.

And then we used the show method, which allows us to show an element that's been hidden. And we used the remove class method to remove a class that's been added to a CSS element. And we also used the hide method that allows us to hide a DOM element that's currently visible in the page.

So unbinding event handlers is something that—it's not really that unusual. It's something that can definitely come up. In this case, it was because we wanted to give the user the ability to kind of restart and undo some things that they already did in the page.

But the unbind method is super powerful, allowing you to remove existing event handlers for one or more DOM elements on the page. Make sure you understand everything we covered in this chapter because coming up, we're going to dive a bit deeper into the unbind method. And specifically, how to use it when you want to work with hyperlinks.

THE UNBIND METHOD PART TWO: HYPERLINKS

Lesson 2: Managing Events

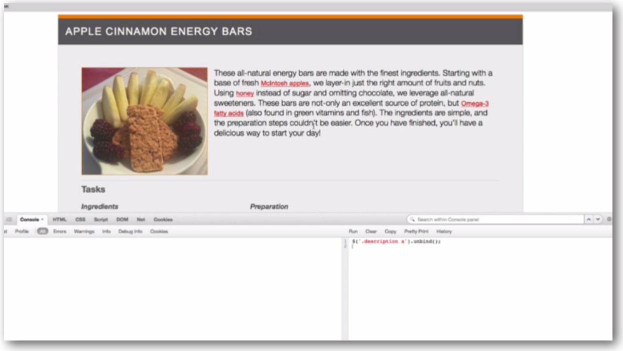
CHAPTER 6

THE UNBIND METHOD PART TWO: HYPERLINKS

GYMNASIUM

JavaScript & jQuery Survival Guide

We've seen how easy it is to remove an existing event handler with the jQuery unbind method. But there are a couple of cases where things can get a little tricky. In your code examples file, copy the code example labeled—attempt to unbind anchor tags. Open your browser and paste this code in the JavaScript console.



APPLE CINNAMON ENERGY BARS

These all-natural energy bars are made with the finest ingredients. Starting with a base of fresh McIntosh apples, we layer-in just the right amount of fruits and nuts. Using honey instead of sugar and omitting chocolate, we leverage all-natural sweeteners. These bars are not-only an excellent source of protein, but Omega-3 fatty acids (also found in green vitamins and fish). The ingredients are simple, and the preparation steps couldn't be easier. Once you have finished, you'll have a delicious way to start your day!

Tasks

Ingredients

Preparation

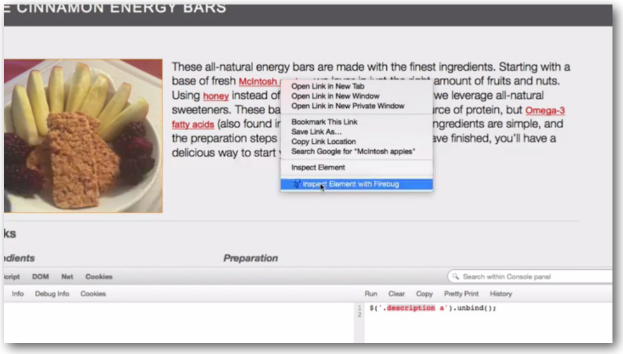
```
$('.description a').unbind();
```

And before we execute it, let's just talk about the page for a moment. In the description paragraph up top, there are three hyperlinks. One says Macintosh apples. One says honey. And one says Omega-3 fatty acids. So each one is a hyperlink that just opens a new browser tab. And it does a Google search on that term.

So if I click Macintosh apples, I can see I've got a Google search for Macintosh apples. And same thing for Omega-3 fatty acids. And same thing for honey. So they're just really simple hyperlinks. And they have, if we inspect them—right mouse button click on one of the hyperlinks and choose Inspect Element with Firebug.

You can see that it has a target attribute whose value equals underscore blank. And that's just what's making it open in a new window. But other than that, they're pretty straight forward. So let's say I want—I don't want these to work. I want to prevent things from working.

So when the user clicks any one of these three anchor tags, I don't want the anchor tags to open a new tab and show the Google search. For now, I want nothing to happen. So we're going to use the unbind method, which we've used successfully before, to remove previous event handlers.



E CINNAMON ENERGY BARS

These all-natural energy bars are made with the finest ingredients. Starting with a base of fresh McIntosh amount of fruits and nuts. Using honey instead of we leverage all-natural sweeteners. These ba rce of protein, but Omega-3 fatty acids (also found i ngredients are simple, and the preparation steps ave finished, you'll have a delicious way to start

ks

dients

Preparation

```
$('.description a').unbind();
```

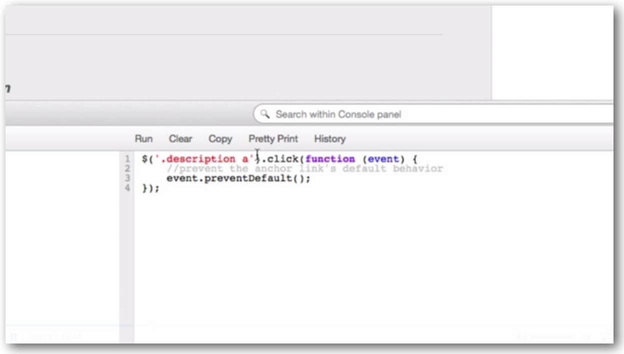
So I'm targeting any anchor tag inside of any element with the class description. Let's just confirm this. So if I click here—Inspect Element with Firebug—I can see that this is an anchor tag inside of an element with the class description.

So that's how we know what we're going to get here. And I'm going to use the unbind method to remove any existing event handlers. So I'm going to run the code, which sets up our event handler. And when I click Macintosh apples, or honey, or Omega-3 fatty acids, nothing should happen.

I don't want these anchor tags to actually work. And that's not what's happening. So when I click honey, it's still opening a new browser tab. When I click Omega-3 fatty acids, it's still opening the browser tab with the Google search. So something's wrong. We're not getting the behavior we expected.

I was hoping that nothing would happen. So something's not right. Well, what's happening is that the unbind method won't work here. It's not what we need because the unbind method only allows you to remove existing event handlers that jQuery manages, or event handlers that you have set up with jQuery.

We haven't set up any event handlers on these anchor tags. We just opened the page. And refreshed the page. And pasted some code in. These anchored tags are doing what comes naturally. They're doing their job. They're executing their default behavior.



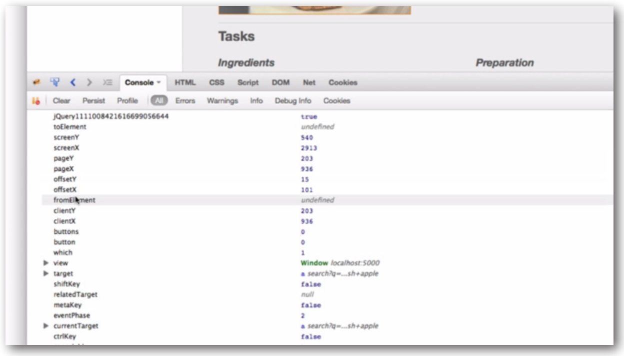
```
$('.description a').click(function (event) {
    //prevent the anchor link's default behavior
    event.preventDefault();
});
```

And that is—when you click them, they open. They take the user to a specified URL or the value of their href attribute. So somehow, we have to override that default behavior. In your code examples file, look for a code example labeled—prevent default behavior of anchor tags.

So copy that code. Refresh the page. Paste that code in. So, same target—anchor tags in the description—click event. But now our event handler has an argument. It's taking argument with the word event. OK. So let's just comment out this line of code first.

And let's learn about this event. So how did we learn about objects, previously? In a previous chapter, we learned how you can inspect an object or learn about an object. And that was the console dot dir method. So if I call console dot dir and I pass it an object, I can learn more about that object.

I happen to know this event—this word event it's representing an object that's going to be passed to our anonymous function. So I think now when I click any one of the anchor tags, I should see an event object appear in my JavaScript console.



```
  jQuery11110084216166990566644          true
  toElement                              undefined
  screenY                                540
  screenX                                2913
  pageY                                  203
  pageX                                  936
  offsetY                                15
  offsetX                                101
  fromElement                            undefined
  clientY                                203
  clientX                                936
  buttons                                0
  button                                 0
  which                                  1
▶ view                                   Window localhost:5000
▶ target                                 a search?q=...sh+apple
  shiftKey                               false
  relatedTarget                          null
  metaKey                                false
  eventPhase                             2
▶ currentTarget                          a search?q=...sh+apple
  ctrlKey                                false
```

So now when I click Macintosh apples, I get that new tab. But now I get a whole bunch of stuff in my JavaScript console—whole bunch of stuff. What do I see—type equals click, time stamp. I see current target. I see offset x, offset y. Those are the coordinates of the mouse when we click the anchor tag.

So this is a whole bunch of information about the event. Think back to our discussion about events like, for example, your friend's birthday. Imagine if that event had an object—an event object. What kind of properties would it have? It would have a day. When's your friend's birthday? It's March 23rd.

It might have a place property. Where is the party? A cake property—chocolate—things like that. So here, the event has all kinds of properties and methods. A lot of them, we're not too interested in right now. But we are interested in one of them—were very interested in.

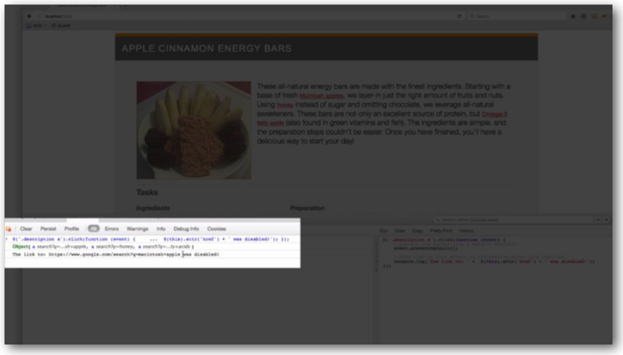
And that is the prevent default method. If you see here, this event object has a prevent default method. We know it's a function and we know we can call it. So let's get rid of the console dir statement. And let's uncomment this line.

Let's refresh the page to get rid of that original event handler. Now, when we execute this code, we see that it acted upon the three anchor length. So it should work. So now when I click these anchor tags, look what happens—nothing happens.

And that's because when each one is clicked, each event handler is getting an event object as an argument. And we're calling that event objects prevent default method. And that's preventing the default behavior of the anchor tag. It's preventing the anchor tag from doing its job.

In this case, it's exactly what we wanted. So, cool—but, why would we do that? It's like we're breaking something that works in the page for the user. Well, let's try to imagine a scenario where we wanted to create this kind of clever feature where a user clicks a hyperlink.

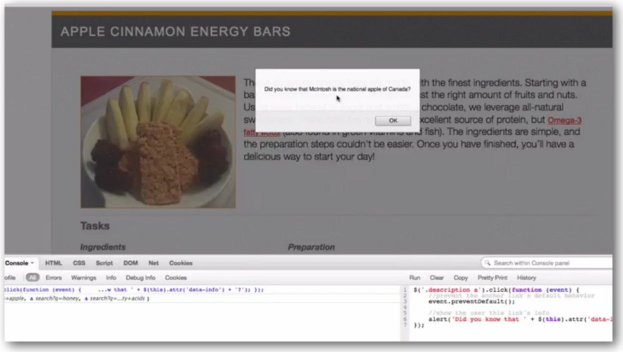
And before we take them to that hyperlink, we want to tell them about the word they've clicked, or a little info pop up. So in your code examples file, go to the code example labeled—view href attribute. And copy that code. Refresh the page. And paste that code into your JavaScript console.



```
> $('.description a').click(function (event) {     ...  $(this).attr('href') + ' was disabled!'); });
Object[ a search?q=...sh+apple, a search?q=honey, a search?q=...ty+acids ]
The link to: https://www.google.com/search?q=macintosh+apple was disabled!
```

Now, run the code. And click one of the hyperlinks. And you'll see, something happens. It says—the link to—and it's the Google search link—was disabled. And if I click honey, I see the same thing. Omega-3 fatty acids—same thing. So it's showing that we have access to information about this hyperlink that we can act upon.

In this case, we're showing the value of the href attribute. We're using we're using the this keyword to get a reference to the element that was clicked. We're using the jQuery attr, or attribute method—it's attr.



```
click(function (event) {     ...w that ' + $(this).attr('data-info') + '?'); });
+apple, a search?q=honey, a search?q=...ty+acids ]
```

```
$('.description a').click(function (event) {
    //prevent the anchor link's default behavior
    event.preventDefault();

    //show the user this link's info
    alert('Did you know that ' + $(this).attr('data-i
});
```

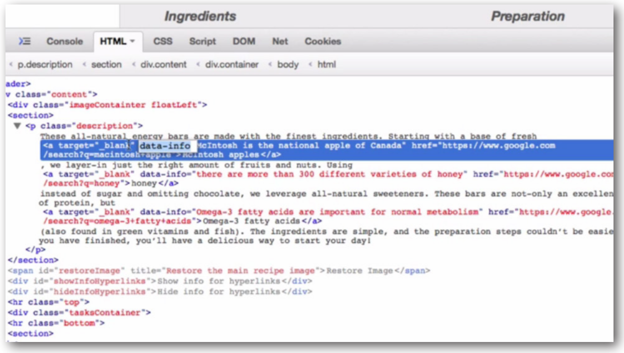
That allows us to get a handle and attribute. And we're saying—hey, I want the href attribute of this element. So that's pretty cool. So we can—when an element is clicked, we can find out about different attributes of that element. And in this case, we're just making a little console statement. It's not very useful.

So if you go back to your examples file, copy the code example labeled—show link info in alert. Refresh the page. And then paste that in your JavaScript console. So now when you run the code and then click one of the anchor links, you'll see—did you know that Macintosh is the national apple of Canada?

Or, click honey—did you know that there are more than 300 different varieties of honey? So what we're doing is, we're saying—hey, when any anchor tag is clicked, we want to prevent the default behavior of that event. So cancel the anchor tag.

And then we do an alert. And we look for the data info attribute of the element that was clicked. Data info—what is that? Well, right mouse button click one of the hyperlinks. And click Inspect Element With Firebug. And you can see each anchor tag has a data info attribute.

And that attribute is some little extra piece of information, like Macintosh is the national capital of apple of Canada and there are more than 300 different varieties of honey. So we have access to all the attributes of the moment that was clicked.



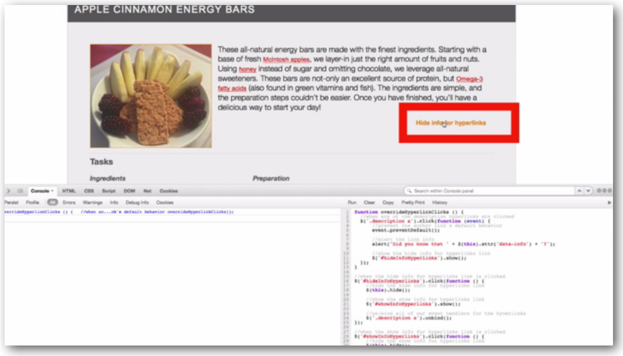
```
ader>
v class="content">
<div class="imageContainter floatLeft">
<section>
 ▼ <p class="description">
     These all-natural energy bars are made with the finest ingredients. Starting with a base of fresh
     <a target="_blank" data-info McIntosh is the national apple of Canada" href="https://www.google.com
     /search?q=macintosh+apple">McIntosh apples</a>
     , we layer-in just the right amount of fruits and nuts. Using
     <a target="_blank" data-info="there are more than 300 different varieties of honey" href="https://www.google.co
     /search?q=honey">honey</a>
     instead of sugar and omitting chocolate, we leverage all-natural sweeteners. These bars are not-only an excellen
     of protein, but
     <a target="_blank" data-info="Omega-3 fatty acids are important for normal metabolism" href="https://www.google.
     /search?q=omega-3+fatty+acids">Omega-3 fatty acids</a>
     (also found in green vitamins and fish). The ingredients are simple, and the preparation steps couldn't be easie
     you have finished, you'll have a delicious way to start your day!
   </p>
</section>
<span id="restoreImage" title="Restore the main recipe image">Restore Image</span>
<div id="showInfoHyperlinks">Show info for hyperlinks</div>
<div id="hideInfoHyperlinks">Hide info for hyperlinks</div>
<hr class="top">
<div class="tasksContainer">
<hr class="bottom">
<section>
```

And we're saying—hey, get the data info attribute of whatever element that was clicked. So it allows us to have this clever little pop up that says—hey, did you know this or did you know that about the thing that you clicked? So that's pretty cool.

So our feature's kind of being flushed out, here. Were preventing the anchor tag from doing what it normally does. We're preventing the link from opening. And then we're giving them a little extra information. So what if we wanted to take it a step further and give people the ability to turn this off and on?

Somebody may say like—look, that's great. I don't care. I just want to click the Macintosh apples link and I want to see the Google search. OK. So if you go your code examples file and copy the code that's labeled—enable show hide info for hyperlinks.

And then refresh the page. Paste that code in here. And then run the code. Now, when you click any of the hyperlinks, we get the info pop up that we expect. But all of a sudden, you see this link—hide info for hyperlinks. OK. So if I clicked that, then, now the links work as normal.



These all-natural energy bars are made with the finest ingredients. Starting with a base of fresh McIntosh apples, we layer-in just the right amount of fruits and nuts. Using honey instead of sugar and omitting chocolate, we leverage all-natural sweeteners. These bars are not-only an excellent source of protein, but Omega-3 fatty acids (also found in green vitamins and fish). The ingredients are simple, and the preparation steps couldn't be easier. Once you have finished, you'll have a delicious way to start your day!

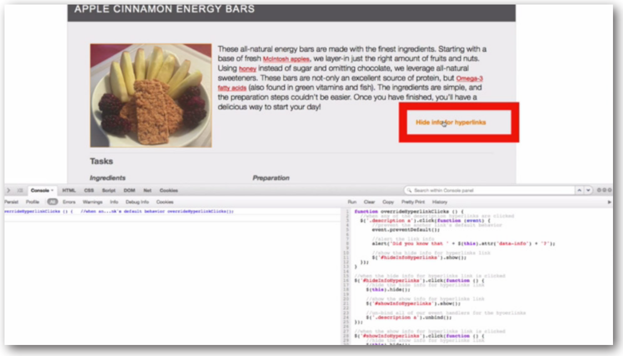
```
overrideHyperlinkClicks () {   //when an...nk's default behavior overrideHyperlinkClicks();
```

```
function overrideHyperlinkClicks () {
  //when any of the description hyperlinks are clicked
  $('.description a').click(function (event) {
      //prevent the anchor link's default behavior
      event.preventDefault();

      //alert the link info
      alert('Did you know that ' + $(this).attr('data-info') + '?');

      //show the hide info for hyperlinks link
      $('#hideInfoHyperlinks').show();
  });
}

//when the hide info for hyperlinks link is clicked
$('#hideInfoHyperlinks').click(function () {
    //hide the hide info for hyperlinks link
    $(this).hide();

    //show the show info for hyperlinks link
    $('#showInfoHyperlinks').show();

    //un-bind all of our event handlers for the hyperlinks
    $('.description a').unbind();
});

//when the show info for hyperlinks link is clicked
$('#showInfoHyperlinks').click(function () {
    //hide the show info for hyperlinks link
```

Now they do exactly what we expect them to do. They open the hyperlink in the new tab. But now it says, show info for hyperlink. So when I click show info for hyperlinks, now we get the info pop up again. So basically, we're toggling the feature.

We're allowing the user to say—I like these info pop ups. I like that when I click a hyperlink I get a little extra information. And then the user might say—all right, I've had enough of that. And I want to hide the info for hyperlinks. And now I want the hyperlinks to work as usual.

So the thing that's kind of cool here is—we've set up a way that we can easily turn the feature on and off. We're literally preventing the default behavior. Or kind of unbinding these anchor tags, and rebinding them over and over, at will, very easily.


APPLE CINNAMON ENERGY BARS

These all-natural energy bars are made with the finest ingredients. Starting with a base of fresh McIntosh apples, we layer-in just the right amount of fruits and nuts. Using honey instead of sugar and omitting chocolate, we leverage all-natural sweeteners. These bars are not-only an excellent source of protein, but Omega-3 fatty acids (also found in green vitamins and fish). The ingredients are simple, and the preparation steps couldn't be easier. Once you have finished, you'll have a delicious way to start your day!

Hide info for hyperlinks

Tasks

Ingredients

Preparation

```
function overrideHyperlinkClicks () {
  //when any of the description hyperlinks are clicked
  $('.description a').click(function (event) {
      //prevent the anchor link's default behavior
      event.preventDefault();

      //alert the link info
      alert('Did you know that ' + $(this).attr('data-info') + '?');

      //show the hide info for hyperlinks link
      $('#hideInfoHyperlinks').show();
  });
}

//when the hide info for hyperlinks link is clicked
$('#hideInfoHyperlinks').click(function () {
    //hide the hide info for hyperlinks link
    $(this).hide();

    //show the show info for hyperlinks link
    $('#showInfoHyperlinks').show();

    //un-bind all of our event handlers for the hyperlinks
    $('.description a').unbind();
});

//when the show info for hyperlinks link is clicked
$('#showInfoHyperlinks').click(function () {
    //hide the show info for hyperlinks link
```

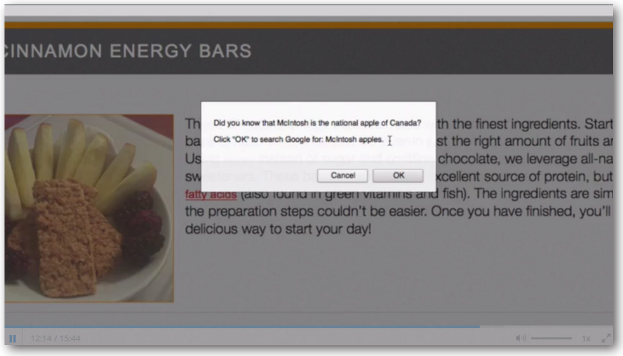
And it really happens with the event objects prevent default method that unbinds, or prevents the default behavior of the anchor tags. And then this show and hide info for hyperlinks links—what happens for each one is when it's clicked, it hides itself. And then it shows the other link. So show info—hide info.

And then it unbinds the anchor tags. So it removes this special event handler that we created up here and allows the hyperlink to work as it normally would. And then when we want to say show info for hyperlinks—it hides itself. It shows the other one.

And then it overrides the hyperlink clicks, which is the first event handler, which is to stop the anchor tag from working. So it's just kind of like a circle, where this feature will work, and then not work, and work, and then not work. And the thing that's really kind of making that happen is this prevent default method.

So let's take this one step further and allow the user to actually do the Google search. So we want to keep the functionality that we have now. We want them to be able to override the anchor links and get that info pop ups. We want them to be able to turn that feature off.

We also want them to be able to actually see the Google search, or follow the href value. So refresh the page. Go back to your code examples file. And copy the code example that's labeled—allow the user to follow the hyperlink if desired. So I'm going to copy all this code. It's kind of a lot.


CINNAMON ENERGY BARS

Did you know that McIntosh is the national apple of Canada?

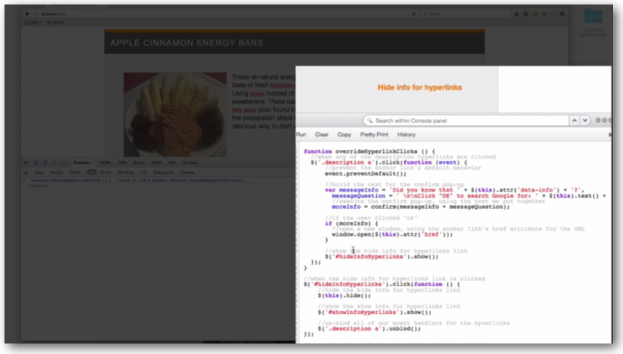
Click "OK" to search Google for: McIntosh apples.

And then paste that code in JavaScript console. Run the code. OK. So now when I click a hyperlink, I get the info pop up. But I also get—click OK to search Google for Macintosh apples. I click OK. I get my Google search. It follows the hyperlink.

And if I click Omega-3—did you know that Omega-3 fatty acids are important for normal metabolism? Great. I get the info. But I also get the option to say—click OK to search Google for Omega-3 fatty acids. And then I can search that.

And if I click another one, and I click—OK, I got the info pop up. But now I don't want to follow the hyperlink so I click Cancel. And nothing happens. I don't get the Google search. I can do the Google search, or I can just forget the Google search.

And I can also turn this feature off. So I can just allow the hyperlinks to work as they normally do. Or I can turn the feature back on and get this info pop up with the option to do the Google search. So how are we doing all that?



```
function overrideHyperlinkClicks () {
  //when any of the description hyperlinks are clicked
  $('.description a').click(function (event) {
      //prevent the anchor link's default behavior
      event.preventDefault();

      //build the text for the confirm pop-up
      var messageInfo = 'Did you know that ' + $(this).attr('data-info') + '?',
        messageQuestion = ' \n\nClick "OK" to search Google for: ' + $(this).text() +
        //execute the confirm pop-up, using the text we put together
        moreInfo = confirm(messageInfo + messageQuestion);

      //if the user clicked "ok"
      if (moreInfo) {
        //open a new window, using the anchor link's href attribute for the URL
        window.open($(this).attr('href'));
      }

      //show the hide info for hyperlinks link
      $('#hideInfoHyperlinks').show();
  });
}

//when the hide info for hyperlinks link is clicked
$('#hideInfoHyperlinks').click(function () {
    //hide the hide info for hyperlinks link
    $(this).hide();

    //show the show info for hyperlinks link
    $('#showInfoHyperlinks').show();

    //un-bind all of our event handlers for the hyoerlinks
    $('.description a').unbind();
});
```

Well, the confirm method is a JavaScript function you can call that allows you to create a little pop-up window that asks the user to basically answer a yes or no question. So when I click this—this is called a confirm. And it's really a yes or no question. It takes one argument—the text that you want the user to see.

In this case, it's—did you know that info about the link? And click OK to do a Google search. So that's the text that's being passed here. We're creating that text right up here. And then the confirm returns a positive or negative value, depending on what the user did.

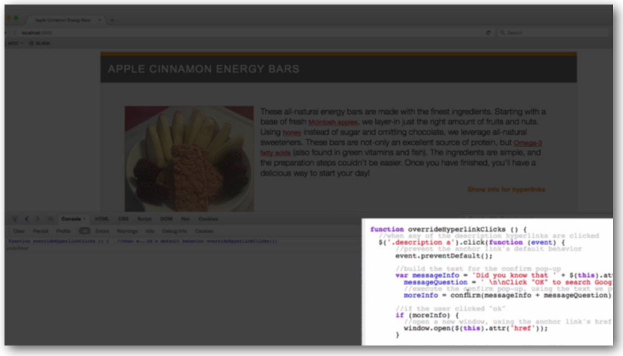
So if they click OK, it returns a positive value. If they click Cancel, it returns a negative value. So that's being set to this more info variable. So if more info means—if they clicked OK, then we're going to say—window dot open. Window is the global object. And it has a method called open.

If we say window dot open and pass it a URL, it will open a new window or a new browser tab with that URL That URL is the href attribute of the element that was clicked.

So that's the reason why I walked through those steps of learning about the attributes of the elements that were clicked because in this case, we're using that href value. It's important to us because we want to say—oh, well they wanted it to Google search, so we need to use that value.

So this confirm is just a really easy way of being able to say—hey, do you want it to do Google search? And if they do, then we open a new window using the href attribute of the anchor tag that was clicked. And if we turn that all off, then the anchor tags just work as normal.

So it's starting to become a lot of code here. Our event handlers are getting more and more sophisticated. But we're doing things that are going to come up. You're going to find yourself wanting to do things like this as time goes on as you're doing front end development because sometimes you want to create event handlers.



```
function overrideHyperlinkClicks () {
  //when any of the description hyperlinks are clicked
  $('.description a').click(function (event) {
      //prevent the anchor link's default behavior
      event.preventDefault();

      //build the text for the confirm pop-up
      var messageInfo = 'Did you know that ' + $(this).at
        messageQuestion = ' \n\nClick "OK" to search Goog
        //execute the confirm pop-up, using the text we p
        moreInfo = confirm(messageInfo + messageQuestion)

      //if the user clicked "ok"
      if (moreInfo) {
        //open a new window, using the anchor link's href
        window.open($(this).attr('href'));
      }
```

Sometimes you want to manually trigger events. Sometimes you want to unbind events or unbind elements, removing existing event handlers. The unbind method is perfect for that. But when you have an anchor tag, the key to unbinding it is to actually use the event object and to prevent that anchor tag's default behavior.

If you want to stop an anchor tag from doing what it would naturally do, you can do that with the prevent default method. In the next chapter, you're going to continue exploring how prevent default can be useful. And we'll be taking a look at the challenges we face when using this method with web forms see you soon.

UNDERSTANDING FORMS AND THE SERVER

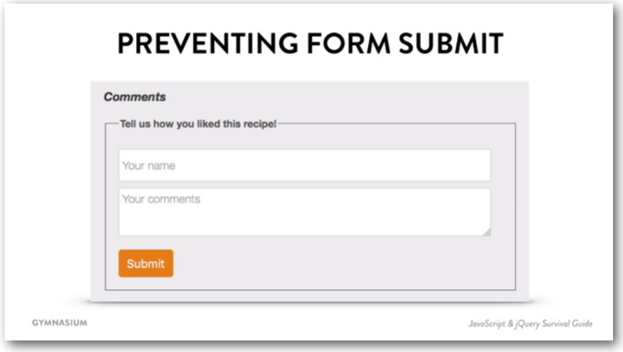
Lesson 2: Managing Events

CHAPTER 7

UNDERSTANDING FORMS AND THE SERVER

GYMNASIUM JavaScript & jQuery Survival Guide

This chapter will be a little bit different in that, we're going to take a quick break from coding and talk about how forms work and how JavaScript fits into the process. Then in Chapter 8, we'll return to coding and apply some of the concepts we learned in this chapter.


PREVENTING FORM SUBMIT

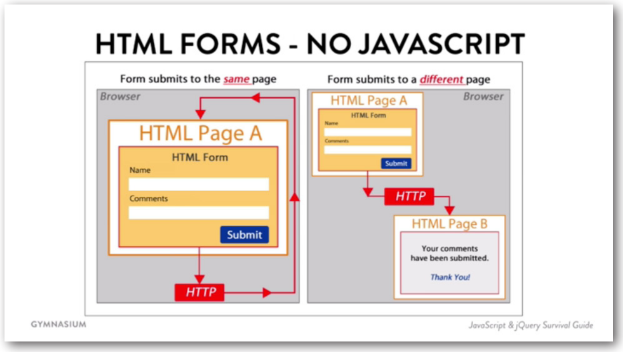
Comments
Tell us how you liked this recipe!
Your name
Your comments
Submit
GYMNASIUM
JavaScript & jQuery Survival Guide

There's another scenario where preventing default behavior is a little tricky. And that's HTML forms. Now, you may have noticed at the bottom of our example page, there's a comments form that allows a user to enter their name and some comments and tell us how they like the recipe.

Well, that's great. And HTML forms have been around since about 1995. It's a very old technology, in the relative scheme of web pages. It works great. It works without JavaScript. It's pure HTML. And it's still used today. And it's terrific. It's fairly simple to implement.

The problem is that the default behavior of HTML forms is that, in one way or another, the user will experience a full page refresh. Either the existing page they're on will be refreshed or they will go to a new page. Let me explain further what I mean here.

Here we have two different scenarios where an HTML form is submitted with no JavaScript. This is the way it worked in 1995 and it can still work this way today if you want to. On the left side, we have one scenario where the form is submitted to the same page.


HTML FORMS - NO JAVASCRIPT

Form submits to the same page
Browser
HTML Page A
HTML Form
Name
Comments
Submit
HTTP
Form submits to a different page
Browser
HTML Page A
HTML Form
Name
Comments
Submit
HTTP
HTML Page B
Your comments have been submitted.
Thank You!
GYMNASIUM
JavaScript & jQuery Survival Guide

And when I say submitted, I mean, well, you've got a form. You've got some information. In this case, you've got name and comments. That information needs to get someplace. So I could, when the form is submitted, I could have that form submit the information to the same exact page.

So you're on page A and you stay on page A. The page is just reloaded. But the information is kind of sent back to the page. And then possibly there's some code in the background that takes that information and acts upon it. So that's scenario one, where you're on the same page, you submit the form, and the page is refreshed.

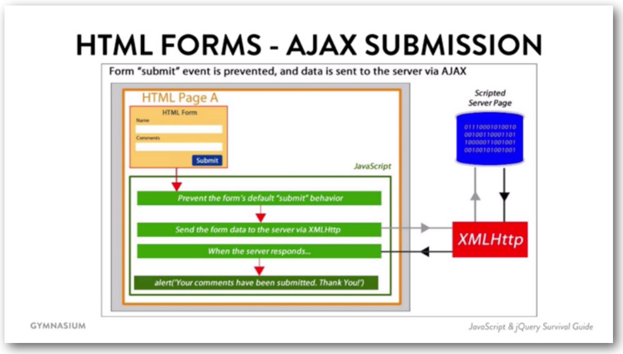
Scenario 2, on the right side, would be when you go from one page to another. So you're starting on page A. You fill out the form. You click Submit. And that information from the form is sent via HTTP to page B. To a completely different page.

So you've got a new page that reloads in the browser. There may be some scripting code in the background that acts upon that data. And then you see some kind of message saying, your comments have been received, thank you very much.

So in both—it doesn't matter which scenario you're in. In either case, there's some kind of round trip to the server. You're either on the same page and it refreshes, or you're on page A and you go to page B. In either case, your page is reloaded and there's going to be some kind of delay.

Now, this is—on paper it's all fine. But the problem is that now a days, users—they've grown accustomed to a slightly more sophisticated experience. Where you're not waiting for that full page refresh. Where things happen on the same page and they happen kind of quickly.

And you're told that things are happening. So let's kind of dive into the more modern experience that I think users have grown accustomed to when it comes to form submission.


HTML FORMS - AJAX SUBMISSION
Form "submit" event is prevented, and data is sent to the server via AJAX
HTML Page A
HTML Form
Name
Comments
Submit
JavaScript
Prevent the form's default "submit" behavior
Send the form data to the server via XMLHttp
When the server responds...
alert('Your comments have been submitted. Thank You!')
Scripted Server Page
XMLHttp
GYMNASIUM
JavaScript & jQuery Survival Guide

Starting around 2005, a term became very popular called AJAX. And now, technically, it stands for Asynchronous JavaScript and XML. And when you have things like Google Maps, where you stay on a page and all kinds of things happen, it utilizes AJAX.

And AJAX relies heavily on something called XMLHTTP. And that's a protocol that allows a web page to communicate with a server, in real time, in the background, without reloading the page. So in this scenario, everything that I'm about to walk through happens and the page never reloads. And you never leave that page.

So you're starting on page A. You fill out the form. You submit it. Right away, some JavaScript is used to prevent that form's default submit behavior. So when the form is submitted, we use the event object and we prevent default and say don't submit the form.

Then we take that data and we send it via XMLHTTP protocol to some server—wherever that server is. And that server may run some code that acts upon that information.

Perhaps you've included some payment information and the server wants to send to the bank, and verify your credit card information, and process some payment, or something like that. And then the server responds and says everything's OK. I got the information everything's OK.

When the server responds, then our page can say, oh, OK, great. Your comments have been submitted. Thank you. Or your payment was received. Or whatever it may be. But the point is, all this happens on page A and there's never a page B. You never leave the page and the page is never refreshed.

And that is—when you use applications such as Facebook, or Twitter, or LinkedIn, or Gmail, this is the kind of experience that you're having in almost every case. You may not consciously think about it but that's exactly what's happening. You stay on the same page.

And when you request things—like when you send an email, receive an email, or send payment, or say that your friends with someone, or you want to look at someone's timeline. In most cases, everything happens behind the scenes.

It's sent to the server. A server responds. And then some action is taken or the web page is updated in real time. So let's go to our example page. And we'll dive into the code and see how we can prevent our comment form from doing its default behavior. And create kind of like more of a Web 2.0 AJAX form submission type of experience.

MIMIC AN AJAX FORM SUBMISSION PART ONE

Lesson 2: Managing Events

CHAPTER 8

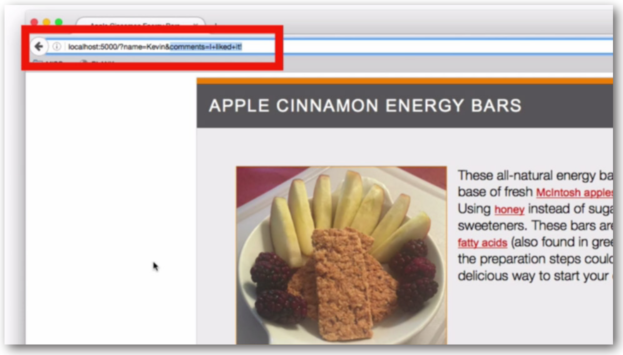
MIMIC AN AJAX FORM SUBMISSION PART ONE

GYMNASIUM

JavaScript & jQuery Survival Guide

Now, let's put the theories from the last chapter into practice. Open up your browser and scroll down to the comments form that you see. Now, before we do anything, I want you to pay close attention to the address bar. Right now, it says local host Colin 5000. That's it.

When you click the Submit button, notice what happens to the URL and the address bar. So I'm going to click Submit. And I see that something has changed. The URL now has what's called query string. And a query string is when you see a question mark with a number of key value pairs added to it. In this case, there's only keys.


localhost:5000/?name=Kevin&comments=I+liked+it!

APPLE CINNAMON ENERGY BARS

There's no values because we didn't fill out the form. Its name equals nothing and comments equal nothing. So if I filled out the form—if I put in my name and I say—I liked it. And then I click submit. Watch what happens to the address bar now.

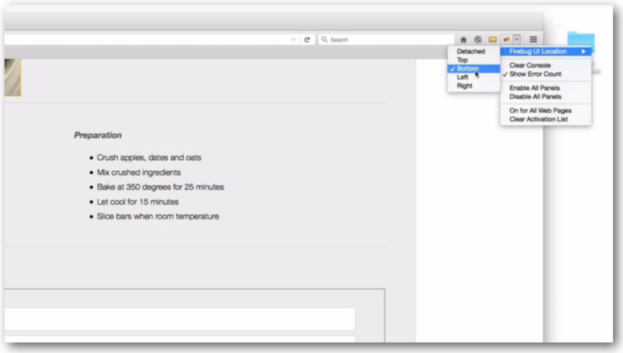
Now there is really a key value pair set. So I have name equals Kevin and comments equals I liked it. So what happened was—the form submitted back to the same page. And this is the scenario I talked about earlier, the left side scenario.

Where the form is submitted to the exact same page. I want to illustrate this a little bit further by using the net tab in Firebug. If you click net, you'll see all of the assets that load when a page is rendered. So if I reload the page, these are all the assets that come over the wire.

It's the actual HTML page itself, the CSS sheet, jQuery, JavaScript library, another JavaScript file, the main image, you can see another image. So when we reload the page you can see these assets coming over. So when you now click the Submit button, watch what happens in the net panel.

You see all those assets. This is just proving that submitting the form is reloading the page. And this is exactly the kind of behavior that we want to prevent. So go back to JavaScript console. And what I want to do before I go into the code is I want to move the location of the Firebug console.

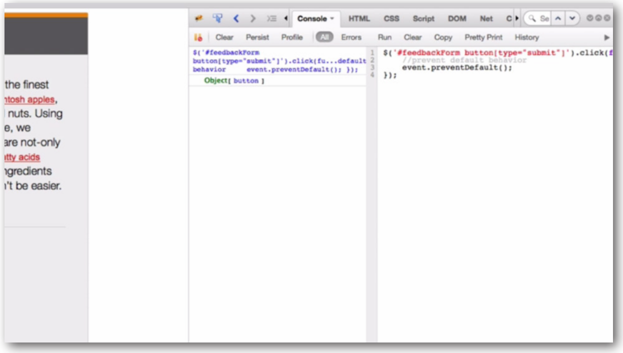
So in the upper right hand corner, click the little down arrow to the right of the Firebug icon. And click Firebug UI Location Right. And I'm going to just zoom out a little bit on the form. So now I've got my JavaScript console positioned so that I can really see the page without having to constantly scroll up and down.



Preparation

- Crush apples, dates and oats
- Mix crushed ingredients
- Bake at 350 degrees for 25 minutes
- Let cool for 15 minutes
- Slice bars when room temperature

Now go to your code examples file and copy the code example labeled—prevent the form's default submit behavior. Paste that code into your JavaScript console. And run the code. Now—In fact, I'm going refresh the page one more time so we can see that the URL is just local host Colin 5000.



```
$('#feedbackForm
button[type="submit"]').click(fu...default
behavior     event.preventDefault(); });
   Object[ button ]
```

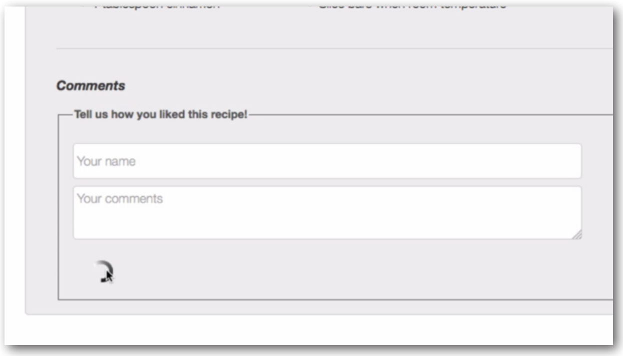
```
$('#feedbackForm button[type="submit"]').click(f
    //prevent default behavior
    event.preventDefault();
});
```

I'm going to run the code. When I click Submit, look what happens. Nothing happens. If you look at the address bar, nothing is happening. In fact, if you go to the net panel and I clear the that panel and they click Submit, I can see that nothing's happening.

The form's not being submitted. So that's great. We've accomplished our first task. We did that by, once again, using the prevent default method of the event object that's passed to our event handler. So preventing the form from being submitted. That's great.

So let's take it to the next step. We want to create, possibly, a better user experience than we would normally experience. So copy the code example labeled—hide submit button show AJAX spinner. And here, we're moving towards kind of like that more modern experience I talked about before, where we don't want the page to refresh.

We don't want to go to a new page. You want to stay on the same page and kind of do things behind the scenes. So if you paste the code in your JavaScript console and you run the code, watch what happens when you click the Submit button.



The Submit button disappears and we see an AJAX spinner. Now, you've seen this before. You've seen these kind of graphics where you're using a page like Facebook, or Twitter, or something, where you do something and then you see a graphic like this, kind of a spinning graphic. And then something appears.

So this AJAX spinner is just a way of leading the user know something's happening. It's just kind of informing the user, something's going on. Hang tight. We'll be done in a second. And I made this Submit button disappear because well, they've just submitted the form.

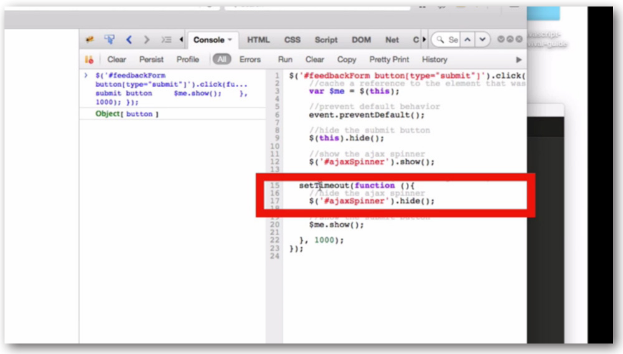
I don't want them to submit it again. I want to prevent them from doing anything else and inform them that something's going on. So I said prevent default. So prevent the form from being submitted. I hide the element that was clicked. I'm using the this keyword to reference the element that was clicked.

In this case, it was a Submit button. And then I'm showing this element with the ID of AJAX spinner. And that's this element, right here. So refresh the page. Go back to your code examples file. And copy a code example labeled—add control reset.

So copy that code. Paste it in the console. Run it. And then, now you click the Reset—the Submit button, you'll notice that it goes away for a second. We see the AJAX spinner. And the Submit button comes back. So it's kind of like this round trip experience.

We submitted our form. We're waiting. And the form has been submitted. And the Submit button's back. And we can go over and over. But we're kind of creating this round trip experience where we're undoing the Submit button, letting the user know something's going on, and then it's over and the Submit button's back.

So that's cool. But we need to actually tell the user the form has been submitted. Now, I just want to also mention that I've use something called setTimeout. It's a JavaScript function that allows you to determine that something should be delayed.



```
> $('#feedbackForm
  button[type="submit"]').click(fu...
  submit button      $me.show();     },
  1000); });
  Object[ button ]
```

```
$('#feedbackForm button[type="submit"]').click(
    //cache a reference to the element that was
    var $me = $(this);

    //prevent default behavior
    event.preventDefault();

    //hide the submit button
    $(this).hide();

    //show the ajax spinner
    $('#ajaxSpinner').show();

  setTimeout(function (){
    //hide the ajax spinner
    $('#ajaxSpinner').hide();

    $me.show();

  }, 1000);
});
```

So what I'm doing is passing the setTimeout function—a function. And saying this function should happen in 1,000 milliseconds. It's calculated milliseconds. So 1,000 milliseconds is a second. And 5,000 milliseconds would be five seconds.

So if I were to refresh the page and change this 1,000 to 5,000, run the code, and then click the Submit button. You'll notice that it took five seconds for it to happen. So the setTimeout method just allows you to say—hey, I want to do something, but I want to wait a little bit.

And I want it to happen in a certain amount of time. So I said—in 5,000 milliseconds, or in five seconds, I want you to execute this function. And this function just simply hides the AJAX spinner and shows a Submit button.

So that's the round trip experience. But we still have a little more work to do. The next chapter will be part two of working with this form. And you'll be learning how to mimic an AJAX form submission. If that doesn't make sense right away, don't worry. I'll walk you through the steps.

MIMIC AN AJAX FORM SUBMISSION PART TWO

In this chapter, we're going to extend the behavior of the form we've been working on. What we'd like to do is use some jQuery in order to give the user useful feedback, such as providing error messages if they don't fill out certain fields, for example. Additionally, we're going to do all this without having to actually submit requests to the server. In some senses, we're going to be faking it. Let's look at how.

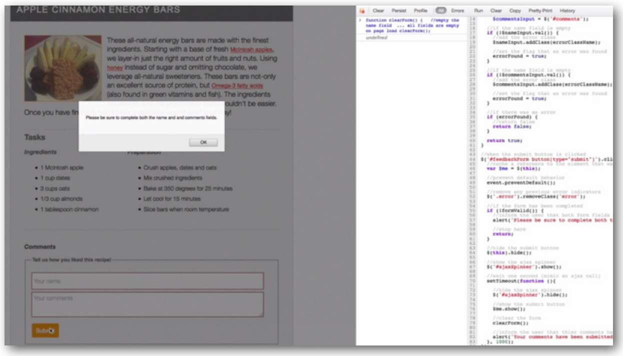
Lesson 2: Managing Events

CHAPTER 9

MIMIC AN AJAX FORM SUBMISSION PART TWO

GYMNASIUM

JavaScript & jQuery Survival Guide



```
function clearForm() {   //empty the
name field  ... all fields are empty
on page load clearForm();
undefined
```

```
      $commentsInput = $('#comments');

    //if the name field is empty
    if (!$nameInput.val()) {
      //add the error class
      $nameInput.addClass(errorClassName);

      //set the flag that an error was found
      errorFound = true;
    }

    //if the name field is empty
    if (!$commentsInput.val()) {
      //add the error class
      $commentsInput.addClass(errorClassName);

      //set the flag that an error was found
      errorFound = true;
    }

    //if there was an error
    if (errorFound) {
      //return false
      return false;
    }

    return true;
  }

  //when the submit button is clicked
  $('#feedbackForm button[type="submit"]').cli
    //cache a reference to the element that wa
    var $me = $(this);

    //prevent default behavior
    event.preventDefault();

    //remove any previous error indicators
    $('.error').removeClass('error');

    //if the form has been completed
    if (!formValid()) {
      //inform the user that both form fields
      alert('Please be sure to complete both t

      //stop here
      return;
    }

    //hide the submit button
    $(this).hide();

    //show the ajax spinner
    $('#ajaxSpinner').show();

    //wait one second (mimic an ajax call)
    setTimeout(function (){

      //hide the ajax spinner
      $('#ajaxSpinner').hide();

      //show the submit button
      $me.show();

      //clear the form
      clearForm();

      //inform the user that thier comments ha
      alert('Your comments have been submitted
    }, 1000);
```

First, refresh your page if you haven't already. And then open up your code examples page and locate this [AUDIO OUT]. And now let's see what happens. I'm going to click Submit. And I get an alert that says, please be sure to complete both the Name and Comments fields. So it's telling me, hey, I can't submit a form that's empty. You're not giving me any information.

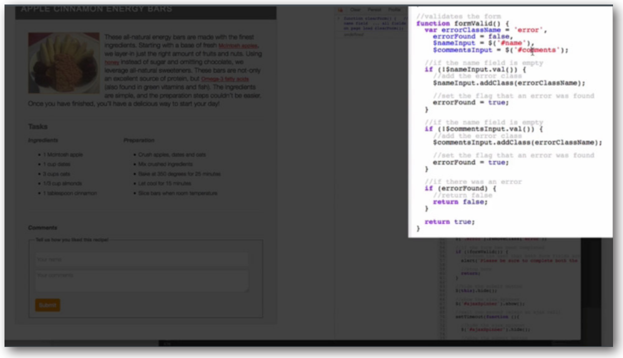
And it not only alerted me that, it changed the UI so that each form has a red border, which kind of indicates an error. So I'm going to fill out the Name field. And then I'm going to submit the form. And it says, please be sure to complete both Name and Comments fields. OK. Because the comments are empty. And I can see that name's OK. Name no longer has this red border or error condition, and Comments has the red border.

So I'm going to say, I loved it, and I'm going to click Submit. And then watch what happens. AJAX spinner, and then we say, hey, your comments have been submitted. Thank you. And we get the Submit button back. And we can kind of delay that. If I refresh the page and I want to say, this should be three seconds that we wait, run the code. I'm going to fill out the form. And submit the form. And that'll take three seconds.

The AJAX spinner's spinning. And then we say, hey, your comments have been submitted. Thank you. And we're pretending that we're talking to the server. And right now it's like, we submit, and it's talking to the server, send it via XML, HTTP. Server responds. It's been submitted. Thank you.

So let's just walk through the code a little bit. We've got this function called clear form. And we're using another jQuery method called val. And the jQuery val method allows you to set the value of an input. It works on inputs.

So I've got this element with the ID of Name. That's this element here. And I've got another element with the ID of Comments. That's this text area here. So in both of them, I'm saying make the val, or value, an empty string. That's what makes the inputs empty. It resets them.

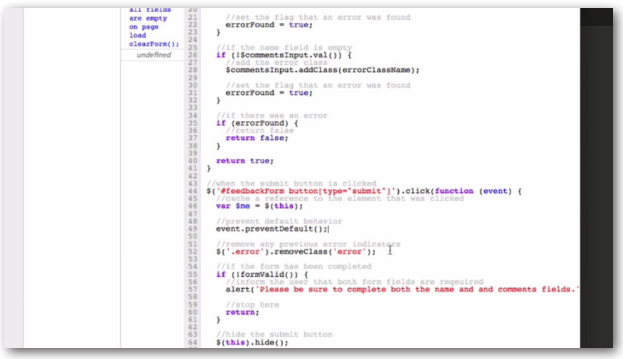


```
//validates the form
function formValid() {
  var errorClassName = 'error',
    errorFound = false,
    $nameInput = $('#name'),
    $commentsInput = $('#comments');

  //if the name field is empty
  if (!$nameInput.val()) {
    //add the error class
    $nameInput.addClass(errorClassName);

    //set the flag that an error was found
    errorFound = true;
  }

  //if the name field is empty
  if (!$commentsInput.val()) {
    //add the error class
    $commentsInput.addClass(errorClassName);

    //set the flag that an error was found
    errorFound = true;
  }

  //if there was an error
  if (errorFound) {
    //return false
    return false;
  }

  return true;
}
```

Then I have this function called form valid, which validates the form. And I'm first getting a reference to the Name and Comments input fields. And then I'm saying, if Name input val is empty, then I'm going to add—I'm going to use the add class method to add a class called error to that input. And that's how we get the red line around the elements. We're adding a class.

In fact, if I expect this element and look at it, you can see that element has a class called error. And if I get rid of that class, the red line disappears. So we're using the add class method to add a class called error to the element that was left empty. If Name is empty, add the error class to Name. If Comments is empty, add the error class to Comments.

And in both cases, if that happens, I set a flag to a variable called error found to equal true. And if error is found, this function returns false. That will mean something to us in a minute. This function returns either true or false. And it's important.



```
all fields
are empty
on page
load
clearForm();
undefined
```

```
    //set the flag that an error was found
    errorFound = true;
  }

  //if the name field is empty
  if (!$commentsInput.val()) {
    //add the error class
    $commentsInput.addClass(errorClassName);

    //set the flag that an error was found
    errorFound = true;
  }

  //if there was an error
  if (errorFound) {
    //return false
    return false;
  }

  return true;
}

//when the submit button is clicked
$('#feedbackForm button[type="submit"]').click(function (event) {
  //cache a reference to the element that was clicked
  var $me = $(this);

  //prevent default behavior
  event.preventDefault();

  //remove any previous error indicators
  $('.error').removeClass('error');

  //if the form has been completed
  if (!formValid()) {
    //inform the user that both form fields are requuired
    alert('Please be sure to complete both the name and and comments fields.'

    //stop here
    return;
  }

  //hide the submit button
  $(this).hide();
```

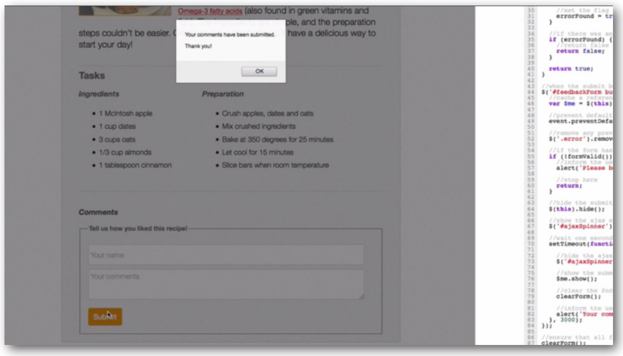
So next up, we have the click handler for this submit. We've seen this before. The first thing we do is we prevent the default behavior of the form, so the form isn't submitted. Then we say any element with the class error, remove that class error. Remove class. We've seen that before.

So just to say, if there was an error before, remove the red lines around text boxes. Let's try this again. Then I say, if not, form valid. So if form valid returns false, that's where we get the alert that says, please be sure to complete both the Name and Comments fields. So that's why it's important if form valid either returns false or true.

And if it returns false, than we show the alert. Otherwise, we return. If it returned false, we stop right here. If it doesn't return false, we continue on, hide the Submit button, show the AJAX spinner. We've been here before. Then we set a timeout, we wait three seconds. And then after three seconds, we say, hide the AJAX spinner, show the Submit button, clear the form, and say, hey, your comments have been received. Thanks very much.

It's during this time that we're kind of pretending that we're talking to a server. I didn't want to spend time talking about the implementation code that would be required to do a real AJAX call to a server. It would kind of be outside of the context of this chapter. But what I wanted to focus on is the front end code that would be required to create that more modern AJAX-based form submission experience.

So in this case, once again, we're going to fill out the form. And then when we click Submit, the page will not refresh. We're not going to go to a different page. We're going to stay on the same page. Then over the course of the next three seconds, we're going to update the user and say, something's happening by showing them the AJAX spinner, pretend that we talked to the server. The server responds.



Omega-3 fatty acids (also found in green vitamins and … ple, and the preparation steps couldn't be easier. … have a delicious way to start your day!

Your comments have been submitted.

Thank you!

Tasks

Ingredients	*Preparation*
• 1 McIntosh apple	• Crush apples, dates and oats
• 1 cup dates	• Mix crushed ingredients
• 3 cups oats	• Bake at 350 degrees for 25 minutes
• 1/3 cup almonds	• Let cool for 15 minutes
• 1 tablespoon cinnamon	• Slice bars when room temperature

Comments

Tell us how you liked this recipe!

```
    //set the flag
    errorFound = tr
  }

  //if there was an
  if (errorFound) {
    //return false
    return false;
  }

  return true;
}

//when the submit b
$('#feedbackForm bu
  //cache a referen
  var $me = $(this)

  //prevent default
  event.preventDefa

  //remove any prev
  $('.error').remov

  //if the form has
  if (!formValid())
    //inform the us
    alert('Please b

    //stop here
    return;
  }

  //hide the submit
  $(this).hide();

  //show the ajax s
  $('#ajaxSpinner')

  //wait one second
  setTimeout(functi

    //hide the ajax
    $('#ajaxSpinner

    //show the subm
    $me.show();

    //clear the for
    clearForm();

    //inform the us
    alert('Your com
  }, 3000);
});

//ensure that all f
clearForm();
```

And when the server responds, we're going to remove the AJAX spinner and then say, hey, we got your comments. Thank you very much. And click Submit. We get the AJAX spinner. And then we see the alert, your comments have been submitted, thank you.

So it's during these three seconds that we would have been talking to the server. Again, I didn't want to spend time talking about that code. But on the front end, we've really kind of learned what we need to do in order to create a more modern, or a kind of Web 2.0 AJAX-based form submission experience by preventing the default behavior, showing an AJAX spinner, and then when the form submission is complete, let the user know we got their comments and clear out the forms.

So old school HTML forms are great. They work great. But nowadays, people have become more accustomed to a little bit more of a sophisticated experience. And the key to that experience is that the page doesn't refresh and you don't leave the page. You stay on the page. Things happen in the background. And then you're updated in real time. And a lot of the methods we've learned about in this chapter have been put to use to make this experience possible.

We covered a lot of material in those last two chapters, so good work. And we've also reached the end of this lesson. In the final lesson, we're going to explore a number of practical and pretty cool ways to control the appearance of your pages with JavaScript, including how to dynamically add text and images to your pages, how to add basic animations, and a lot more. I'll see you soon.