HOW EVENT DELEGATION AFFECTS THE $(THIS) OBJECT

As mentioned earlier, the `$(this)` object refers to the element that's currently being handled within a loop (page 139) or by an event handler (page 159). Normally, in an event handler, `$(this)` refers to the initial selector. For example:

```
$('ul').on('click', function() {
  $(this).css('text-decoration': 'line-through');
}
```

In the above code, `$(this)` refers to the `<ul>` tag the visitor clicks. However, when you use event delegation, the initial selector is no longer the element that's being interacted with—it's the element that contains the element the visitor clicks on (or mouses over, tabs to, and so on). Take a look at the event delegation code one more time:

```
$('ul').on('click', 'li', function() {
  $(this).css('text-decoration': 'line-through');
}); // end on
```

In this case, the `<li>` tag is the one the visitor will click, and it's the element that needs to respond to the event. In other words, `<ul>` is just the container, and the function needs to be run when the `<li>` tag is clicked. So, in this example, `$(this)` is going to refer to the `<li>` tag, and the function above will add a line through each `<li>` tag the visitor clicks.

For many tasks, you won't need event delegation at all. However, if you ever need to add events to HTML that isn't already on the page when it loads, then you'll need to use this technique. For example, when you use Ajax (Chapter 13), you may need to use event delegation to apply events to HTML content that's dynamically added to a web page from a web server.

NOTE In some cases, you may want to use event delegation simply to improve JavaScript performance. If you're adding lots and lots of tags to the same event handler, for example, hundreds of table cells in a large table, it's often better to delegate the event to the `<table>` tag like this:

```
$('table').on('click', 'td', function () {
  // code goes here
});
```

By adding the event to the table, you're avoiding having to apply event handlers directly to hundreds or even thousands of individual elements, a task that can consume browser memory and processing power.

Tutorial: A One-Page FAQ

"Frequently Asked Questions" pages are a common sight on the Web. They can help improve customer service by providing immediate answers 24/7. Unfortunately, most FAQ pages are either one very long page full of questions and complete answers, or

a single page of questions that link to separate answer pages. Both solutions slow down the visitors' quest for answers: in the first case, forcing a visitor to scroll down a long page for the question and answer he's after, and in the second case, making the visitor wait for a new page to load.

In this tutorial, you'll solve this problem by creating a JavaScript-driven FAQ page. All of the questions will be visible when the page loads, so it's easy to locate a given question. The answers, however, are hidden until the question is clicked—then the desired answer fades smoothly into view (Figure 5-7).

Overview of the Task

The JavaScript for this task will need to accomplish several things:

- When a question is clicked, the corresponding answer will appear.
- When a question whose answer is visible is clicked, then the answer should disappear.

FREQUENTLY ASKED QUESTION

Is Event Delegation Voodoo?

OK, I understand the basics of event delegation, but how does it actually work?

As you read in the box on page 167, event "bubbles" up through the HTML of a page. When you click on a link inside a paragraph, the `click` event is first triggered on that link; then, the parent element—the paragraph—gets the `click` event, followed by the `<body>` and then the `<html>`. In other words, an event that's triggered on one HTML element passes upwards to each of its parents.

This fact can be really helpful in the case of event delegation. As mentioned on page 171, sometimes you want to apply events to HTML that doesn't yet exist—like an item in a to-do list—that only comes into existence after the page loads and a visitor adds a list item. Although you can't add a `click` event to a `<li>` tag that isn't there yet, you can add the `click` event to an already existing parent like a `<ul>` tag, or even a `<div>` tag that holds that `<ul>` tag.

As discussed on page 164, every event has an event object that keeps track of lots of different pieces of information. In this case, the most important piece of information is the `target` property. This property specifies the exact HTML tag receiving the event. For example, when you click a link, that link is the target of the click. Because of event bubbling, a parent tag can "hear" the event and then determine which child element was the target.

So, with event delegation, you can have a parent element listen for events—like a `<ul>` tag listening for a `click` event. Then, it can check to see which tag was the actual target. For example, if the `<li>` tag was the target, then you can run some specific code to handle that situation—like crossing off the list item as in the to-do list example.

In addition, you'll want to use JavaScript to hide all of the answers when the page loads. Why not just use CSS to hide the answers to begin with? For example, setting the CSS `display` property to `none` for the answers is another way to hide the answers. The problem with this technique is what happens to visitors who don't have JavaScript turned on: They won't see the answers when the page loads, nor will they be able to make them visible by clicking the questions. To make your pages

viewable to both those with JavaScript enabled and those with JavaScript turned off, it's best to use JavaScript to hide any page content.

NOTE See the note on page 12 for information on how to download the tutorial files.

The Programming

1. **In a text editor, open the file *faq.html* in the *chapter05* folder.**

   This file already contains a link to the jQuery file, and the `$(document).ready()` function (page 160) is in place. First, you'll hide all of the answers when the page loads.

2. **Click in the empty line after the `$(document).ready()` function, and then type `$('.answer').hide();`.**

   The text of each answer is contained within a `<div>` tag with the class of `answer`. This one line of code selects each `<div>` and hides it (the `hide()` function is discussed on page 184). Save the page and open it in a web browser. The answers should all be hidden.

NOTE The elements are hidden with JavaScript instead of CSS, because some visitors may not have JavaScript turned on. If that's the case, they won't get the cool effect you're programming in this tutorial, but they'll at least be able to see all of the answers.

   The next step is determining which elements you need to add an event listener to. The answer appears when a visitor clicks the question, therefore you must select every question in the FAQ. On this page, each question is a `<h2>` tag in the page's main body.

3. **Press Return to create a new line and add the code in bold below to the script:**

```
<script src="../_js/jquery.min.js></script>
<script>
$(document).ready(function() {
  $('.answer').hide();
 $('.main h2')
}); // end of ready()
</script>
```

   That's a basic descendant selector used to target every `<h2>` tag inside an element with a class of `main` (so it doesn't affect any `<h2>` tags elsewhere on the page). Now it's time to add an event handler. The `click` event is a good candidate, but you'll need to do something more to make it work. For this example, each click will either show the answer or hide it. That sounds like a job for a conditional statement. Essentially, you want to check whether the answer `<div>` that's placed after the `<h2>` tag is hidden: if it is, then show it, if not, then hide it.

4. **Add the following bolded code to attach an event handler to the <h2> tags:**

```
$(document).ready(function() {
  $('.answer').hide();
 $('.main h2').click(function() {

 }); // end click
}); // end of ready()
```

This code adds a `click` event to those `<h2>` tags. The `// end click` comment isn't required, but, as mentioned on page 58, a comment can really help you figure out what code a set of `});` characters belongs to. With this code in place, whatever you put inside the anonymous function here will run each time a visitor clicks one of those `<h2>` tags.

5. **On the empty line inside the function, type:**

```
var $answer = $(this).next('.answer');
```

Here, you're creating a variable—`$answer`—that will hold a jQuery object. As discussed on page 159, `$(this)` refers to the element currently responding to the event—in this case, a particular `<h2>` tag. jQuery provides several functions to make moving around a page's structure easier. The `.next()` function finds the tag immediately following the current tag. In other words, it finds the tag following the `<h2>` tag. You can further refine this search by passing an additional selector to the `.next()` function—the code `.next('.answer')` finds the first tag following the `<h2>` that also has the class `answer`.

In other words, you're storing a reference to the `<div>` immediately following the `<h2>` tag, That `<div>` holds the answer to the question. You're storing it in a variable, because you'll need to access that element several times in the function: to see whether the answer is hidden, to show the answer if it is hidden, and to hide it if it's visible. Every time you access jQuery using the `$( )`, you're telling the browser to fire off a bunch of programming code inside jQuery. So the code `$('this').next('.answer')` makes jQuery do some work. Instead of repeating those same steps over and over, you can just store the results of that work in a variable, and use that variable again and again to point to the `<div>` you wish to hide or show.

When you need to use the same jQuery results over and over and over again, it's a good idea to store the results a single time in a variable you can use repeatedly. This makes the program more efficient, saves the browser from having to unnecessarily do extra work, and make your web page more responsive.

The variable begins with a `$` symbol (as in `$answer`) to show that you're storing a jQuery object (the result of running the `$( )` function). It's not a requirement that you add the `$` in front of the variable name; it's just a common convention jQuery programmers use to let them know that they can use all the wonderful jQuery functions—like `.hide()`—with that variable.

NOTE The `.next()` function is just one of the many jQuery functions (also called *methods*) that help you navigate through a page's DOM. To learn about other helpful functions, visit *http://docs.jquery.com/Traversing*. You can also read more in "Traversing the DOM" on page 531.

6. **Add an empty if/else clause to your code, like this:**

```
$(document).ready(function() {
  $('.answer').hide();
 $('.main h2').click(function() {
  var $answer = $(this).next('.answer');
  if ( ) {

  } else {

  }
 }); // end click
}); // end of ready()
```

Experienced programmers don't usually type an empty if/else statement like this, but as you're learning it can be really helpful to build up your code one piece at a time. Now, why not test to see whether the answer is currently hidden.

7. **Type `$(answer.is(':hidden')` inside the parentheses of the conditional statement:**

```
$(document).ready(function() {
 $('.answer').hide();
$('.main h2').click(function() {
 var $answer = $(this).next('.answer');
 if ($answer.is(':hidden')) {

 } else {

 }
 }); // end click
}); // end of ready()
```

You're using the `$answer` variable you created in step 5. That variable contains the element with the class `answer` that appears immediately after the `<h2>` tag the visitor clicks. Remember, that's a `<div>` containing the answer to the question inside the `<h2>` tag.

You're using jQuery's `is()` method to see whether that particular element is hidden. The `is()` method checks to see whether the current element matches a particular selector. You give the function any CSS or jQuery selector, and that function tests whether the object matches the given selector. If it does, the function returns a `true` value; if not, it returns a `false` value. That's perfect! As you know, a conditional statement needs either a `true` or `false` value to work (page 66).

The `:hidden` selector used here is a special, jQuery-only selector that identifies hidden elements. In this case, you're checking to see whether the answer is currently hidden. If not, then you can show it.

8. **Add line 6 below to your program:**

```
$(document).ready(function() {
  $('.answer').hide();
 $('.main h2').click(function() {
  var $answer = $(this).next('.answer');
  if ($answer.is(':hidden')) {
   $answer.slideDown();
  } else {

  }
 }); // end click
}); // end of ready()
```

The `slideDown()` function is one of jQuery's animation functions (you'll learn more about animation in the next chapter). It reveals a hidden element by sliding it down onto the page. At this point, you can check out your hard work. Save the page and check it out in a web browser. Click one of the questions on the page. The answer below it should open (if it doesn't, double-check your typing and refer to the troubleshooting tips on page 18).

If you look at the page, you'll see a blue + symbol to the left of each headline. The plus sign is a common icon used to mean, "Hey, there's more here." To indicate that a visitor can click to hide the answer, replace the plus sign with a minus sign. You can do it easily by adding a class to the `<h2>` tag.

9. **After the line of code you added in the last step, type the following:**

```
$(this).addClass('close');
```

Remember that `$(this)` applies to the element that's receiving the event (page 159). In this case, that's the `<h2>` tag. Thus, this new line of code adds a class named `close` when the answer is shown. The minus sign icon is defined within the style sheet as a background image. (Once again, CSS makes JavaScript programming easier.)

In the next step, you'll complete the second half of the toggling effect—hiding the answer when the question is clicked a second time.

10. **In the `else` section of the conditional statement, add two more lines of code (bolded below). The finished code should look like this:**

```
<script src="../_js/jquery.min.js></script>
<script>
$(document).ready(function() {
  $('.answer').hide();
 $('.main h2').click(function() {
  var $answer = $(this).next('.answer');
  if ($answer.is(':hidden')) {
   $answer.slideDown();
   $(this).addClass('close');
  } else {
   $answer.fadeOut();
   $(this).removeClass('close');
  }
 }); // end click
}); // end of ready()
</script>
```

This part of your program hides the answer. You could have used the `slideUp()` function, which hides the element by sliding it up and out of view, but to add interest and variation, in this case you'll fade the answer out of view using the `fadeOut()` function (about which you'll learn more on page 185).

Finally, you'll remove the `close` class from the `<h2>` tag: this makes the + sign reappear at the left of the headline.

Save the page and try it out. Now when you click a question, not only does the answer appear, but the question icon changes (Figure 5-7).

TIP Once you're done, try replacing the `slideDown()` function with `fadeIn()` and the `fadeOut()` function with `slideUp()`. Which of these animation functions do you prefer?

FIGURE 5-7
With just a few lines of JavaScript, you can make page elements appear or disappear with a click of the mouse.