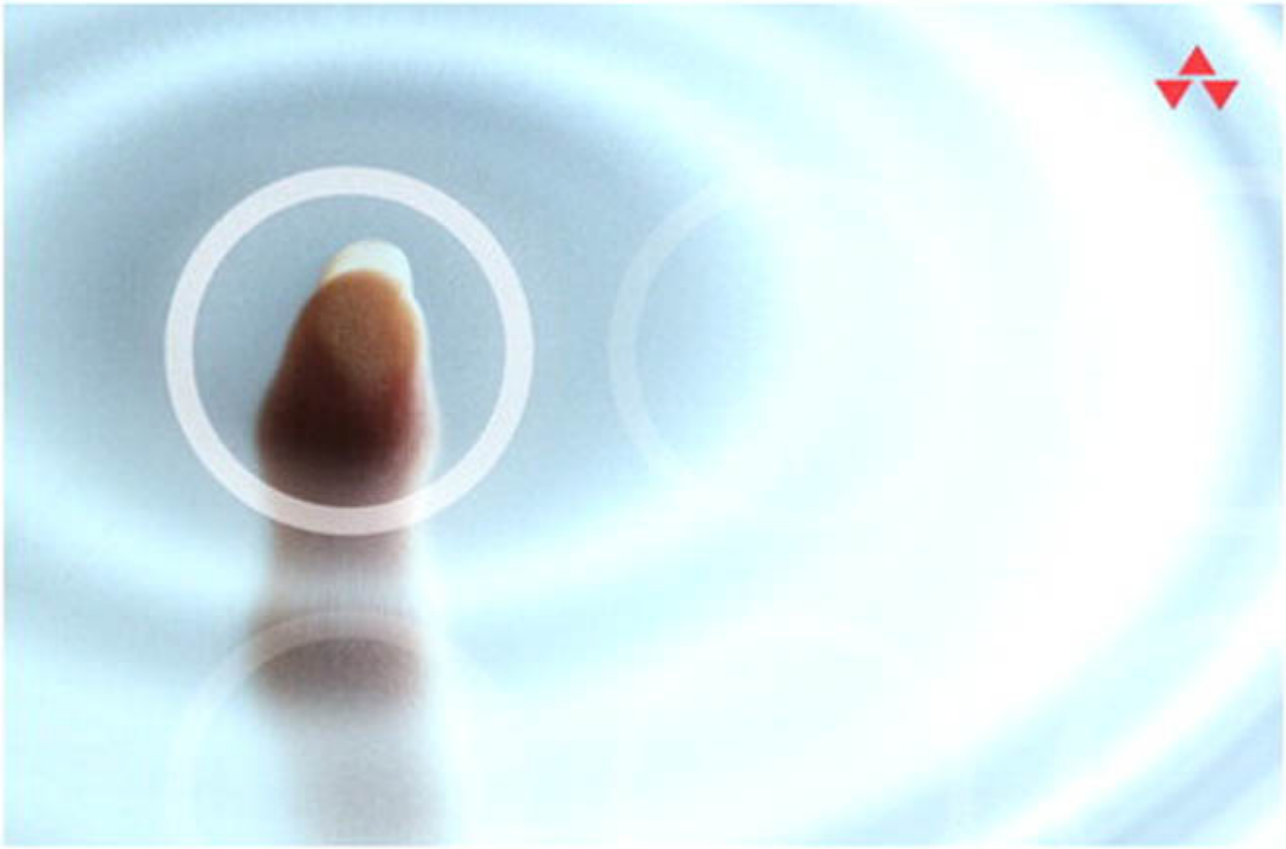


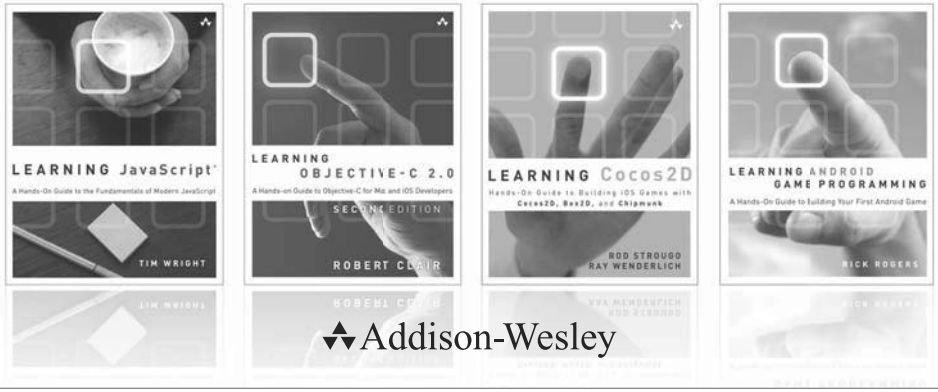
LEARNING jQuery

A Hands-on Guide to Building Rich Interactive Web Frontends

Learning jQuery

A Hands-on Guide to Building
Rich Interactive Web Front Ends

Addison-Wesley Learning Series



Visit informit.com/learningseries for a complete list of available publications.

The Addison-Wesley Learning Series is a collection of hands-on programming guides that help you quickly learn a new technology or language so you can apply what you've learned right away.

Each title comes with sample code for the application or applications built in the text. This code is fully annotated and can be reused in your own projects with no strings attached. Many chapters end with a series of exercises to encourage you to reexamine what you have just learned, and to tweak or adjust the code as a way of learning.

Titles in this series take a simple approach: they get you going right away and leave you with the ability to walk off and build your own application and apply the language or technology to whatever you are working on.

Learning jQuery

A Hands-on Guide to Building Rich Interactive Web Front Ends

Ralph Steyer

Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Cape Town • Sydney • Tokyo • Singapore • Mexico City

Learning jQuery: A Hands-on Guide to Building Rich Interactive Web Front Ends

ISBN-13: 978-0-321-81526-2
ISBN-10: 0-321-81526-2

Library of Congress Cataloging-in-Publication Data is on file.

Printed in the United States of America

First Printing May 2013

Trademarks

Warning and Disclaimer

Bulk Sales

Editor in Chief
Mark Taub

Acquisitions Editor
Mark Taber

Managing Editor
Sandra Schroeder

Project Editor
Mandie Frank

Copy Editor
Keith Cline

Indexer
Larry Sweazy

Proofreader
Megan Wade

Translator
Almut Dworak

Technical Editor
Brad Dayley

Publishing Coordinator
Vanessa Evans

Designer
Chuti Prasertsith

Compositor
Jake McFarland

Contents at a Glance

Table of Contents

About the Author

Ralph Steyer is a computer programmer, consultant, journalist, and book author with decades of experience in a wide variety of computer programming languages and technologies. He has a degree in mathematics from Frankfurt/Main University and is the author of several books on web programming, including *JavaScript Handbook* and *AJAX Frameworks* (Addison-Wesley).

We Want to Hear from You!

As the reader of this book, *you* are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

You can email or write directly to let us know what you did or didn't like about this book—as well as what we can do to make our books stronger.

Please note that we cannot help you with technical problems related to the topic of this book, and that due to the high volume of mail we receive, we might not be able to reply to every message.

When you write, please be sure to include this book's title and author, as well as your name and phone or email address.

Email:	feedback@developers-library.info
Mail:	Reader Feedback Addison-Wesley Developer's Library 800 East 96th Street Indianapolis, IN 46240 USA

Reader Services

Visit our website and register this book at www.informit.com/register for convenient access to any updates, downloads, or errata that might be available for this book.

Your purchase of this book includes access to a free online edition for 45 days through the Safari Books Online subscription service. Details are on the last page of this book.

This page intentionally left blank

2

First Examples with jQuery

In this chapter, we make first contact with jQuery without any further preparations. In other words, we are jumping right into the deep end. I am anxious for you to get a feeling for what you can do with jQuery and what you can get out of this framework. Just accept for now that many questions regarding the source text have to remain open at this stage. Don't worry, though; these questions are answered over the next few chapters. The explanations on the listings also remain somewhat superficial at this stage, to avoid going off topic. We want to get into the practical application of jQuery as quickly as possible and just have some fun playing around, which means creating examples.

> **Note**
>
> For the examples in this chapter, but also most examples in the following chapters, it is not relevant which specific version of jQuery you are using. The examples in this book have been created with jQuery 1.8.2 or later, but often any version from 1.3 or at least 1.4.1 onward is sufficient.

2.1 Accessing Elements and Protecting the DOM

If you already have some basic knowledge of programming on the Web,[1] you already know that you can access the components of a web page via JavaScript or another script language in the browser via an object model with the name Document Object Model (DOM). For this type of access, there are several standard techniques,[2] each of which has its own weaknesses.

1. Given the target audience of this book, I assume you do.

2. For example, the methods `getElementById()` and `getElementsByTagName()` plus access via object fields or names.

In particular, you usually have to enter many characters when accessing just a single element of the web page (or a group). This involves a lot of effort and is susceptible to errors. Most frameworks therefore offer a system via which this access can take place with an abbreviated, unified approach. Plus the underlying mechanisms compensate for various weaknesses of the standard access methods, above all by compensating for browser-dependent particularities and supplementing various missing functions of the pure DOM concept. Particularly important is that this compensation has generally been tested on all officially supported browsers and therefore works rather reliably.

The following example demonstrates another extremely important function of jQuery—protecting the DOM. More on what this is all about later. For now, let's just say that different browsers process the web page differently on loading (parsing) the page, which can lead to a number of problems when the elements of the web page are accessed (especially if you try to access the elements of the web page too soon in a script—in other words, before the browser has correctly constructed the DOM). Here, jQuery offers a reliably method for mastering these problems.

The example also shows you in passing, as it were, how you can use jQuery as a standardized way of accessing contents of elements with text and reacting to events. But enough introduction. Here is our very first listing (ch2_1.html):[3]

Listing 2.1 **The First jQuery Example**

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 01
Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
 <head>
   <meta http-equiv="Content-Type"
     content="text/html; charset=utf-8" />
   <title>The first jQuery example</title>
   <script type="text/javascript"
       src="lib/jquery-1.8.min.js"></script>
   <script type="text/javascript">
     $(document).ready(function(){
       $("#a").click(function(){
         $("#output").html("Boring :-(");
       });
       $("#b").click(function(){
         $("#output").html("A nice game :-)");
       });
       $("#c").click(function(){
```

3. The quotations are from the movie *War Games*—one of the first movies about a hacker. I highly recommend that you watch it the next time it is on TV.

```
18          $("#output").html("A strange game. " +
19               "The only winning move " +
20              "is not to play.");
21        });
22      });
23    </script>
24  </head>
25  <body>
26    <h1>Welcome to WOPR</h1>
27    <h3>Shall we play a game</h3>
28    <button id="a">Tic Tac Toe</button>
29    <button id="b">Chess</button>
30    <button id="c">
31      Worldwide Thermonuclear War</button>
32    <div id="output"></div>
33  </body>
34 </html>
```

Just create the HTML file in a separate directory and save it under the listed name.

In practice, you would usually save all your resources that are part of a project within a separate directory. For a web project, the best solution is to create these directories in the shared folder of your web server. In the case of Apache/XAMPP, this is usually the directory htdocs. This has the advantage that—if the web server is running—you can run the test directly via HTTP and a proper web call, not just load the file via the FILE protocol into the browser (in other words, the classic opening as file or simply dragging the file into the browser). The latter is not a realistic, practice-related test because later the pages also have to be requested by the visitor via a web server.

If you are working with an integrated development environment (IDE) such as Aptana or the Visual Studio Web Developer, you can usually display a web page directly from the IDE via an integrated web server. In Aptana, this is done via the **Run** command, and in Web Developer (a Firefox add-on) you can use the shortcut **Ctrl+F5.**

> **Note**
>
> In this book, all examples are sorted by chapter and listed accordingly on the companion website (http://jquery.safety-first-rock.de).

In lines 7 and 8, you see the reference to an external JavaScript file—the jQuery library that in this specific case resides in the subdirectory lib of the project directory where the website is saved. This structure has now become widely accepted in practice. This means that the jQuery library also has to be located in exactly that place. But, of course, you can instead choose to use a different path structure.

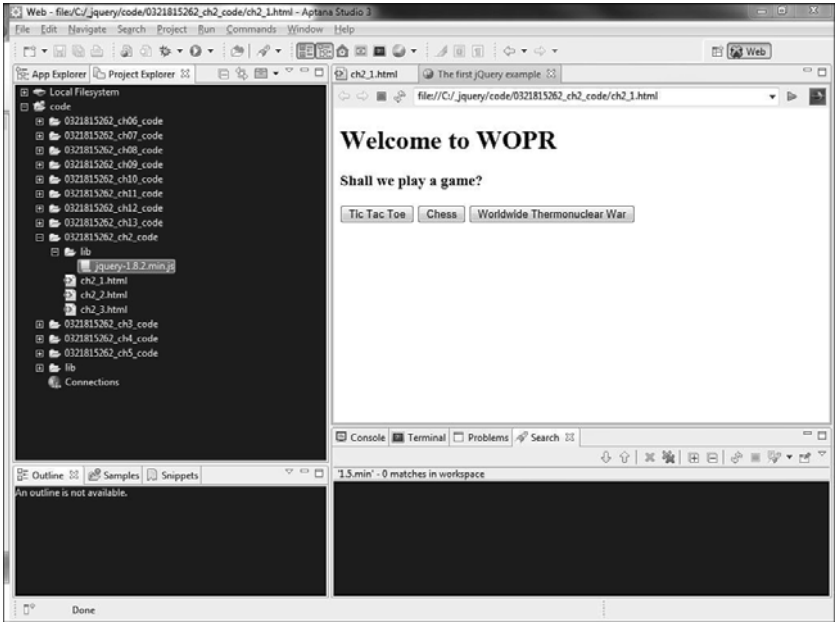


Figure 2.1 In this project, the jQuery library is located in the directory lib, seen from the perspective of the website.

Line 9 to 23 contains a normal JavaScript container. In it, the web page is addressed with `$(document)` (line 10). The function `$()` is a shorthand notation for referencing an element of the web page. You also see these shortened access notations in lines 11, 12, 14, 15, 17, and 18. But here, an element ID is used as a parameter.

> **Note**
>
> Note that an element (in terms of jQuery) as a parameter of `$()` is not enclosed in quotation marks, whereas an ID (or another selector) is enclosed in quotation marks.

Let's now take a quick look at the method `ready()` that starts in line 10 and goes up to line 22. This method ensures that the calls it contains are only executed when the web page has been fully loaded and the DOM is correctly constructed. As hinted at before and without going into too much detail, this is already a feature whose value cannot be appreciated highly enough.

> **Note**
>
> For readers with the corresponding knowledge and experience, the method `ready()` is an alternative for the event handler `onload` that you can write in HTML in the body of a web page or under JavaScript for the corresponding DOM object. But this event handler is seen as extremely *unreliable* because it is insufficiently implemented in various browsers. It is a good idea to avoid it wherever possible.

Within the `ready()` method, three event handlers each specify the reaction when clicking the listed elements. In our examples, these are three buttons marked with a unique ID.

> **Note**
>
> The method `click()` encapsulates (you probably guessed it) the function call of the event handler `onclick`.

The allocation to the correct function is achieved via the ID and triggering the function within the method `click()`. Note that we are using an anonymous function here (without an identifier).

It also gets interesting if a user clicks one of the buttons. This displays a specific text output in a section of the web page. We are again using `$()` and an ID for selecting the section (a `div` block) and the method `html()` for accessing the content.

Figure 2.2 The web page with the three buttons; the user has just clicked the third button.

> **Note**
>
> In all following examples, we omit writing or using the `DOCTYPE` statement. For the sake of completeness, it does belong there, but omitting it does not have any effect for us, and because it is always the same, writing it down over and over again is just a waste of space in this book. In the examples on the companion website, the statement is included because it forms part of the correct standard.

2.2 Editing the Web Page with DHTML à la jQuery

Generally, you can design the visual appearance of a web page much better and more effectively with style sheets than with pure HTML. In particular, they make it easier to separate layout and structure of the site. These statements are probably old hat to you, as true as they are.

If you now change the style sheets of a site dynamically via JavaScript, we are talking about Dynamic Hypertext Markup Language (DHTML). But animation effects such as showing and hiding parts of a web page via other JavaScript techniques also form part of this. In the following example, we look at how you can carry out animated web page changes with jQuery quickly, simply, conveniently, and yet reliably in the various browsers. In this example, we change the Cascading Style Sheets (CSS) class of an element dynamically.

First, let's look at a little CSS file that should be integrated into the following web page and saved in the lib directory (ch2_2.css):

Listing 2.2 **The File with the External Style Sheets**

```
01 body {
02  background: lightgray;color: blue;
03 }
04 div {
05  background: white;font-size: 14px;
06 }
07 .mClass {
08  background: red; color: yellow; font-size: 24px;
09 }
```

Nothing much happens in the CSS file. It determines the background and foreground color of the entire web page and all elements of the type `div`, plus the font size for all elements of the type `div`.

Of primary interest is the class described in lines 7–9. It is not yet to be used on loading the following web page, but is to be assigned dynamically in case of a user action (ch2_2.html):

Listing 2.3 **Changing the CSS Class**

```
01 <html xmlns="http://www.w3.org/1999/xhtml">
02  <head>
03  <meta http-equiv="Content-Type"
04    content="text/html; charset=utf-8" />
05  <title>The second jQuery example</title>
06  <link type="text/css" rel="stylesheet"
07    href="lib/ch2_2.css" />
08  <script type="text/javascript"
09    src="lib/jquery-1.8.2.min.js"></script>
```

```
10  <script type="text/javascript">
11    $(document).ready(function(){
12      $("#a").click(function(){
13        $("#c").addClass("mClass");
14      });
15      $("#b").click(function(){
16        $("#c").removeClass("mClass");
17      });
18    });
19  </script>
20  </style>
21 </head>
22 <body>
23  <h1>Editing Style Sheets with jQuery</h1>
24  <button id="a">Add CSS class</button>
25  <button id="b">Remove CSS class</button><hr/>
26  <div id="c">He who knows all the answers
27    has not been asked all the questions.
28  </div><hr/>
29  <div id="c">Be not afraid of going slowly,
30   be afraid only of standing still.</div>
31 </body>
32 </html>
```

In the example, you can see two buttons below a heading and two texts within a `div` section that is separated by a separator in each case. This is pure HTML. Plus in lines 6 and 7 you can see the link to the CSS file.

Figure 2.3 The site after loading

But now we want to use jQuery to manipulate the text below the buttons or the first `div` container. That is why the `div` container has an ID. The text below it is intended for comparison purposes.

For accessing the elements of the web page, the example uses jQuery mechanisms already mentioned in the first example. To react to the relevant click on a button, we again use the method `click()`. So far, there is nothing new.

Now you should notice that we do not yet assign the CSS class from the linked CSS file to an element on loading the web page. But take a look at line 13.

Listing 2.4 **Adding a CSS Class**

```
$("#c").addClass("mClass");
```

As the name of the method `addClass()` already implies, calling this method assigns the indicated style sheet class to the preceding element. This happens dynamically without the web page having to be reloaded in any way. The function is triggered when the user clicks the corresponding button, as you can see from the surrounding `click()` method.

Figure 2.4 The CSS class has been assigned.

In line 16, you can see how the class is removed again following the same pattern. This time, we use the method `removeClass()`. If you test the example, you will see that font and background are changed accordingly.

> **Tip**
>
> Alternatively, you could use the method `toggleClass()` in this example. This removes or adds a CSS class, always depending on the state. If the class is already assigned, it is then removed, and vice versa.

2.3 Animatedly Reducing and Enlarging of an Element

Now we want to use jQuery to animatedly reduce and enlarge an element to hide or show it. First, let's look at the external CSS file in the subdirectory lib, in which a property is defined that has specific consequences for the following animation (ch2_3.css):

Listing 2.5 **The CSS File**

```
01 body {
02  background: lightgray;color: blue;
03 }
04 #i1 {
05  width:300px; height:225px;
06 }
07 #i2 {
08  height:225px;
09 }
10 #h2{
11  background: white; color:#0000FF; font-size: 18px;
12 }
```

The specification that is interesting for the following example is the width data in line 5. The ID used as selector references an image. The width specification influences the type of the animation that follows. But first, let's look at the web page itself. It basically contains two images and some text below. We want to animate all three elements (ch2_3.html):

Listing 2.6 **Reducing or Enlarging Two Images and Some Text**

```
...
06    <link type="text/css" rel="stylesheet"
07      href="lib/ch2_3.css" />
08    <script type="text/javascript"
09      src="lib/jquery-1.8.min.js"></script>
10    <script type="text/javascript">
11      $(document).ready(function(){
12        $("#toggle1").click(function(event){
13          $('#i1').slideToggle('slow');
14        });
15        $("#toggle2").click(function(event){
16          $('#i2').slideToggle('slow');
17        });
```

```
18        $("#toggle3").click(function(event){
19          $('#h2').slideToggle('slow');
20        });
21      });
22    </script>
23  </head>
24  <body>
25    <h1>Animated showing and hiding
26   of an image and text with jQuery</h1>
27    <button id="toggle1">Toggle Image 1</button>
28    <button id="toggle2">Toggle Image 2</button>
29    <button id="toggle3">Toggle Text</button><hr/>
30    <img src="images/i1.jpg" id="i1" />
31    <img src="images/i2.jpg" id="i2" /><hr/>
32    <h2 id="h2">A ski jump</h2>
33  </body>
34 </html>
```

At the core of this animation is the method `slideToggle()`. This name is also very telling. You can use this effect to show or hide objects depending on the current state, or to reduce or enlarge them. In other words, the current state is toggled to the opposite state. You can see it applied in lines 13, 16, and 19.

Tip

As you can probably see, a temporal interval is specified as a parameter. It determines how long the animation should take. You can pass such parameters for the speed in most animations in jQuery. Permitted parameters are `slow`, `normal`, `fast`, or a specification of time in milliseconds. The specification in milliseconds is not enclosed by quotation marks.

Figure 2.5 The original looks like this.

If you reconstruct the animation of the first image, you will see that reducing the image results in a reduction in image height and the image then disappears altogether. Vice versa, the image grows upward from that point if you enlarge it.

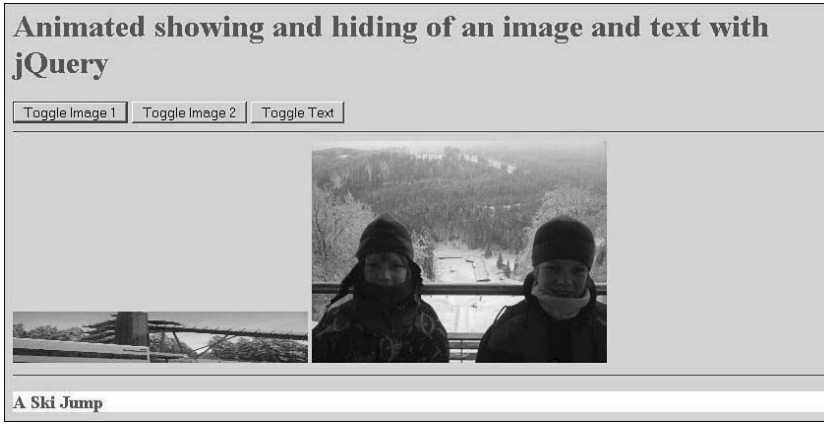


Figure 2.6 Here, the first image is squashed down.

Of massive importance for this behavior is that the width of this image is specified via the CSS rule for the ID `i1`. This prevents the width from also being reduced. The animation of the second image whose width is not specified shows what that looks like. You will see that on reducing the image, it shrinks into the lower-left corner of the image and then disappears altogether. Vice versa, the image grows outward from this point to the top right.

Figure 2.7 The second image shrinks animatedly in width and height.

But now observe what happens to the text if you click the third button. The heading also disappears but only in terms of height.

For the effect of `slideToggle()`, it matters to what type of element the animation technique is applied, and the CSS rules that have been previously applied to an element also play a role.

The animations in the example are basically independent from one another. If you set the interval for running the animation long enough, you can have the animations run in parallel.

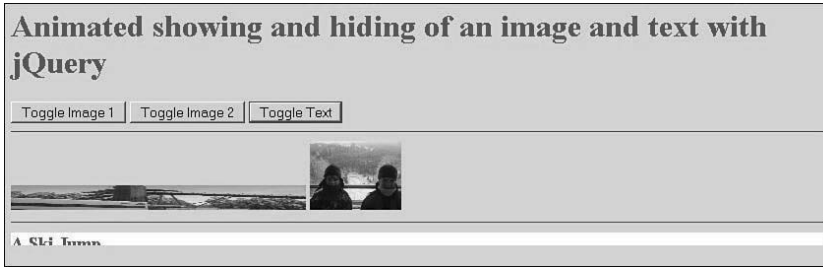


Figure 2.8 The three elements are animated in parallel.

In that case, note that the content positioned lower down is moved upward or could be repositioned vertically if a preceding element has disappeared completely. (In effect, it is removed from the text flow.)

But what happens in the example if you click the same button several times? This answer might surprise you: The events are cumulated. This means they are executed consecutively; the next event is executed only when the previous one has been fully processed. So, clicking the button again does not cause the current animation to stop and the next one to commence immediately. If that is what you want to achieve, you have to program it explicitly.

2.4 Changing Attributes Dynamically

This section shows how you can dynamically change attributes for an element of the web page. To this purpose, jQuery offers the extremely flexible and useful method `attr()`. With this, you can dynamically change one or several attributes of an element. In curly brackets, you set a value pair as parameter, first specifying the attribute, followed by a colon and then a string with the new value. Alternatively, you can also specify two string parameters. In this variation, the first parameter represents the attribute name and the second parameter the value. (In this case, you can change only one attribute.) If you only want to request the value of an attribute, you just enter the name of the attribute as a string parameter.

> **Note**
>
> For the sake of simplicity, we change only one attribute in the following example. If you want to change several attributes at once, however, you just need to write additional value pairs in the curly brackets, separated by commas.

For our example, we want to replace an image in the web page by changing the value of the attribute `src` of an `<img>` tag (ch2_4.html).

Listing 2.7 **Changing Attributes on an Element**

```
...
06    <script type="text/javascript"
07      src="lib/jquery-1.8.min.js"></script>
08    <script type="text/javascript">
09      $(document).ready(function(){
10        $("#toggle1").click(function(){
11          $("img").attr({
12            src: "images/i1.jpg"
13          });
14        });
15        $("#toggle2").click(function(){
16          $("img").attr(
17            "src", "images/i2.jpg"
18          );
19        });
20      });
21    </script>
22  </head>
23  <body>
24    <h1>Replacing an image</h1>
25    <button id="toggle1">Image 1</button>
26    <button id="toggle2">Image 2</button>
27    <hr/><img src="images/i1.jpg"/>
28 </html>
```

We change the value via the notation in the curly brackets once, and via the two string parameters once. As mentioned earlier, we replace the value `src` in each case.

Summary

This chapter provided just a few examples, but they already serve well to demonstrate central key facts of jQuery. In particular, you should memorize the function `$()` and the method `ready()`. But techniques for specifying reactions, such as the method `click()`, are also of elementary importance. And animation techniques such as `addClass()`, `toggleClass()`, `removeClass()`, and `slideToggle()` are also going to be helpful in practice later for DHTML effects. In this chapter, you also learned about changing attribute values (`attr()`). You will more fully understand these techniques as you work through other chapters of this book and have delved deeper into the overall concept of jQuery.

Index

SYMBOLS

A

B

C

D

E

F

G

H

I

J

K

L

M

N

Q

R

S

T

U

V

W

X–Z