



# Learning Center

# Users

There's a lot more to learn about building web sites and applications with jQuery than can fit in API documentation. If you're looking for explanations of the basics, workarounds for common problems, best practices, and how-tos, you're in the right place!

## Authors

Too much good information is spread across corners of the web, languishing in blog and forum posts, often just out of the reach of people who need it, while the same questionable advice is duplicated across even more questionable sites. Help us stem the tide and educate today's — and tomorrow's — web

## Chapters

- [About jQuery](#)
- [JavaScript 101](#)
- [Using jQuery Core](#)
- [Events](#)
- [Effects](#)
- [Ajax](#)
- [Plugins](#)
- [Performance](#)
- [Code Organization](#)
- [jQuery UI](#)
- [jQuery Mobile](#)

## Recently Updated

- [Why Use the Widget Factory](#)
- [How To Use the Widget Factory](#)
- [How To Write a Theme](#)
- [Using jQuery UI ThemeRoller](#)
- [jQuery UI CSS Framework API](#)
- [Widget Factory](#)
- [Theming jQuery UI](#)
- [How jQuery UI Works](#)
- [Getting Started with jQuery UI](#)
- [jQuery UI](#)

## Open Source Content

All of the content in this site is completely open source, and we welcome your contribution. Whether you notice a small improvement that should be made, or want to write entirely new articles, this is one area where feature requests are encouraged!

**Open an Issue or Submit a Pull Request**

Each of our articles has a link to the raw content on GitHub, and we urge everyone to fork, edit, and help improve this

community  
resource!

# About jQuery

Getting started with jQuery can be easy or challenging, depending on your experience with JavaScript, HTML, CSS, and programming concepts in general.

- [How jQuery Works](#)
- [Additional jQuery Support](#)

# How jQuery Works

## jQuery: The Basics

This is a basic tutorial, designed to help you get started using jQuery. If you don't have a test page setup yet, start by creating the following HTML page:

```
1  <!doctype html>
2  <html>
3    <head>
4      <meta charset="utf-8">
5      <title>Demo</title>
6    </head>
7    <body>
8      <a href="http://jquery.com/">jQuery</a>
9      <script src="jquery.js"></script>
10     <script>
11       // Your code goes here
12     </script>
13   </body>
14 </html>
```

The `src` attribute in the `<script>` element must point to a copy of jQuery. Download a copy of jQuery from the [Downloading jQuery](#) page and store the `jquery.js` file in the same directory as your HTML file.

## Launching Code on Document Ready

To ensure that their code runs after the browser finishes loading the document, many JavaScript programmers wrap their code in an `onload` function:

```
1 | window.onload = function() {
2 |     alert("welcome");
3 | }
```

Unfortunately, the code doesn't run until all images are finished downloading, including banner ads. To run code as soon as the `document` is ready to be manipulated, jQuery has a statement known as the [ready event](#):

```
1 | $( document ).ready( function() {
2 |     // Your code here
3 | });
```

For example, inside the `ready` event, you can add a click handler to the link:

```
1 | $( document ).ready(function() {
2 |     $("a").click(function( event ) {
3 |         alert("Thanks for visiting!");
4 |     });
5 | });
```

Save your HTML file and reload the test page in your browser. Clicking the link should now first display an alert pop-up, then continue with the default behavior of navigating to <http://jquery.com/>.

For `click` and most other [events](#), you can prevent the default behavior by calling `event.preventDefault()` in the event handler:

```
1 | $( document ).ready(function() {
2 |     $("a").click(function( event ) {
3 |         alert("As you can see, the link no longer
4 |         event.preventDefault();
```

```
5     });  
6     });
```

## Complete Example

The following example illustrates the click handling code discussed above, embedded directly in the HTML `<body>`. Note that in practice, it is usually better to place your code in a separate JS file and load it on the page with a `<script>` element's `src` attribute.

```
1  <!doctype html>  
2  <html>  
3    <head>  
4      <meta charset="utf-8">  
5      <title>Demo</title>  
6    </head>  
7    <body>  
8      <a href="http://jquery.com/">jQuery</a>  
9      <script src="jquery.js"></script>  
10     <script>  
11       $( document ).ready(function() {  
12         $("a").click(function( event ) {  
13           alert("The link will no longer tak  
14             event.preventDefault();  
15           });  
16         });  
17     </script>  
18 </body>  
19 </html>
```

## Adding and Removing an HTML Class

**Important:** You must place the remaining jQuery examples inside the `ready` event so that your code executes when the document is

ready to be worked on.

Another common task is adding or removing a `class`.

First, add some style information into the `<head>` of the document, like this:

```
1 | <style>
2 |     a.test {
3 |         font-weight: bold;
4 |     }
5 | </style>
```

Next, add the [addClass\(\)](#) call to the script:

```
1 | $("a").addClass("test");
```

All `a` elements are now bold.

To remove an existing `class`, use [removeClass\(\)](#):

```
1 | $("a").removeClass("test");
```

## Special Effects

jQuery also provides some handy [effects](#) to help you make your web sites stand out. For example, if you create a click handler of:

```
1 | $("a").click(function( event ){
2 |     event.preventDefault();
3 |     $( this ).hide("slow");
4 | });
```

then the link slowly disappears when clicked.

## Callbacks and Functions

Unlike many other programming languages, JavaScript enables you to freely pass functions around to be executed at a later time. A *callback* is a function that is passed as an argument to another function and is executed after its parent function has completed. Callbacks are special because they patiently wait to execute until their parent finishes. Meanwhile, the browser can be executing other functions or doing all sorts of other work.

To use callbacks, it is important to know how to pass them into their parent function.

### Callback *without* Arguments

If a callback has no arguments, you can pass it in like this:

```
1 | $.get( "myhtmlpage.html", myCallback );
```

When `$.get` finishes getting the page `myhtmlpage.html`, it executes the `myCallback` function. **Note** that the second parameter here is simply the function name (but *not* as a string and without parentheses).

### Callback *with* Arguments

Executing callbacks with arguments can be tricky.

#### Wrong

This code example will *not* work:

```
1 | $.get( "myhtmlpage.html", myCallback(param1,
```

```
param2 )
```

The reason this fails is that the code executes `myCallback( param1, param2 )` immediately and then passes the `myCallback`'s *return*

*value* as the second parameter to `$.get`. We actually want to pass in `myCallback` the function, not `myCallback`'s return value (which might or might not be a function). So, how to pass in `myCallback` *and* include its arguments?

## Right

To defer executing `myCallback` with its parameters, you can use an anonymous function as a wrapper. Note the use of `function() {`. The anonymous function does exactly one thing: calls `myCallback`, with the values of `param1` and `param2`.

```
1 | $.get( "myhtmlpage.html", function() {  
2 |     myCallback( param1, param2 );  
3 | });
```

When `$.get` finishes getting the page `myhtmlpage.html`, it executes the anonymous function, which executes `myCallback( param1, param2 )`.

# Additional jQuery Support

While we hope to cover most jQuery-related topics on this site, you may need additional or more immediate support. The following resources can prove useful.

## Official Forums

<http://forum.jquery.com/>

There are many subforums where you can discuss jQuery, ask questions, talk about JavaScript, or announce your plugins.

- [Getting Started](#)
  - This is the best place to post if you are brand new to jQuery and JavaScript.
- [Using jQuery](#)
  - This is the best place to post if you have general questions or concerns.
  - If you've built a site that uses jQuery, or would like to announce a new plugin, this is the place to do it.
- [Using jQuery Plugins](#)
  - If you are a plugin author or user and you wish to discuss specific plugins, plugin bugs, new features, or new plugins.
- [Using jQuery UI](#)
  - This is the place to discuss use of [jQuery UI](#) Interactions, Widgets, and Effects
- [jQuery Mobile](#)
  - This is the place to discuss jQuery Mobile.
- [Developing jQuery Core](#)
  - This forum centers around development of the jQuery library itself.
  - Post here if you have questions about certain bugs, development with jQuery, features, or anything in the bug tracker or Git.

- [Developing jQuery Plugins](#)
  - This forum covers development of jQuery plugins.
- [Developing jQuery UI](#)
  - This is the place to discuss development of [jQuery UI](#) itself - including bugs, new plugins, and how you can help.
  - All jQuery UI svn commits are posted to this list to facilitate feedback, discussion, and review.
  - Also note that a lot of the development and planning of jQuery UI takes place on the [jQuery UI Development and Planning Wiki](#).
- [Developing jQuery Mobile](#)
  - This forum covers issues related to the development of jQuery Mobile.
- [QUnit and Testing](#)
  - This is the place to discuss JavaScript testing in general and QUnit in particular

At the bottom of each of the forums is an RSS feed you can subscribe to.

To ensure that you'll get a useful answer in no time, please consider the following advice:

- Ensure your markup is valid.
- Use Firebug/Developer Tools to see if you have an exception.
- Use Firebug/Developer Tools to inspect the html classes, css, etc.
- Try expected resulting html and css without javascript/jquery and see if the problem could be isolated to those two.
- Reduce to a minimal test case (keep removing things until the problem goes away, etc.)
- Provide that test case as part of your mail. Either upload it somewhere or post it on jsbin.com.

In general, keep your question short and focused and provide only essential details - others can be added when required.

## Mailing List Archives

The mailing list existed before the forums were created. The mailing lists were closed in early 2010.

There are two different ways of browsing the mailing list archives.

- 1 The official mailing list archives can be found here:
  - [jQuery General Discussion Archives](#)
  - [jQuery Dev List Archives](#)
  - [jQuery UI General Discussion Archives](#)
  - [jQuery UI Dev List Archives](#)
  - [jQuery Plugins List Archives](#)

Also, an interactive, browsable, version of the General Discussion mailing list can be found on [Nabble](#) (a forum-like mailing list mirror).

## Chat / IRC Channel

jQuery also has a very active IRC channel, **#jquery**, hosted by [freenode](#).

The IRC Channel is best if you need quick help with any of the following:

- JavaScript
- jQuery syntax
- problem solving
- strange bugs.

If your problem is more in-depth, we may ask you to post to the mailing list, or the bug tracker, so that we can help you in a more-suitable environment.

### Connect info:

**Server:** irc.freenode.net

**Room:** #jquery

You can also connect at <http://webchat.freenode.net/?channels=#jquery>.

Additionally we have **#jquery-es** and **#jquery-de** if you want to speak your native language.

If you wish to post code snippets to the channel, you should use a paste site, like <http://jsfiddle.net/> or <http://jsbin.com/>

## StackOverflow

There is an active and well-informed support community at [StackOverflow](#). You can likely find an answer for whatever issue you're experiencing. If your question isn't addressed, you can ask a new question and often receive a quick response.

# JavaScript 101

## Introduction

So you want to unlock the power of jQuery to make the web a better place? Awesome, but there are a few things you should know about JavaScript first.

Introduced at the dawn of the web, [JavaScript](#) is a powerful and expressive language that runs inside the browser in conjunction with HTML and CSS. Based on an open standard called [ECMAScript](#), JavaScript has quickly become the "programming language of the web". All the power of jQuery is accessed via JavaScript, so needless to say, it's an important language to learn. Having a basic knowledge of JavaScript will go a long way in understanding, structuring and debugging your code.

This guide covers the foundational concepts of JavaScript, as well as frequent pitfalls developers fall into during their first foray into the language. When possible, we'll relate the JavaScript you learn here to how it's applied in jQuery.

If you have experience with other programming languages, good for you! If not, don't worry. We'll take it slow and teach you everything you need to know to unlock the power of jQuery with JavaScript.

## Table of Contents

- [Getting Started](#)
- [Running Code](#)
- [Syntax Basics](#)
- [Types](#)
- [Operators](#)
- [Conditional Code](#)
- [Loops](#)
- [Reserved Words](#)
- [Arrays](#)
- [Objects](#)
- [Functions](#)
- [Testing Type](#)
- [The “this” Keyword](#)
- [Scope](#)
- [Closures](#)

# Getting Started

# Anatomy of a Web Page

Before diving into JavaScript, it helps to understand how it aligns with the other web technologies.

## HTML is for Content

HTML is a markup language used to define and describe content. Whether it be a blog post, a search engine result or an e-commerce site, the core content of a web page is written in HTML. A semantic markup, HTML is used to describe content in universal terms (headers, paragraphs, images, etc.).

## CSS is for Presentation

CSS is a supplemental language that applies style to HTML documents. CSS is all about making content look better by defining fonts, colors and other visual aesthetics. The power of CSS comes from the fact that styling is not intermingled with content. This means you can apply different styles to the same piece of content, which is critical when building responsive websites that look good across a range of devices.

## JavaScript is for Interactivity

In the browser, JavaScript adds interactivity and behavior to HTML content. Without JavaScript, web pages would be static and boring. JavaScript helps bring a web page to life.

Look at this simple HTML page that includes CSS and JavaScript to see how it all fits together:

```
1  <!DOCTYPE HTML>
2  <html lang="en-US">
3  <head>
4    <meta charset="UTF-8">
5    <title>Hello World</title>
6    <!-- CSS for presentation -->
```

```
7     <style type="text/css">
8         h1 { font-size: 14px; color: hotpink; }
9         button { color: red; }
10    </style>
11 </head>
12 <body>
13     <h1>Hello World</h1>
14     <button>Click Me!</button>
15
16     <!-- JavaScript for interactivity -->
17     <script>
18         // get a handle on the first button element
19         // in the document.
20         var button = document.querySelector('button'
21         // if a user clicks on it, say hello!
22         button.addEventListener('click', function(ev) {
23             alert('Hello');
24         }, false);
25     </script>
26 </body>
27 </html>
```

In the example above, HTML is used to describe the content. The "Hello World" text is described as a heading with the `<h1>` tag and "Click Me!" is described as a button with the `<button>` tag. The `<style>` block contains CSS that changes the font-size and color of the header text. The `<script>` block contains JavaScript that adds interactivity to the button. When a user clicks on the button, an alert message will appear that says "Hello!".

## A Scripting Language for the Web

JavaScript was originally designed to add interactivity to web pages, not to be a general programming language, which makes it a scripting language. [Scripting languages](#) are regarded to be more productive than general languages because they are optimized for their specific domain (in this case, the web browser). However, recent advancements have brought JavaScript to the server-side (via [Node.js](#)) so it can now be used in place of languages like PHP, Ruby or ASP. This guide will focus exclusively on JavaScript running in the browser with jQuery.

The name "JavaScript" is a bit misleading. Despite the similarity in name, JavaScript has no relationship with [Java](#)), a general purpose language. JavaScript is based on an Open Web standard called [ECMAScript](#). Standards-based languages are not controlled by any one entity or corporation - instead, developers work together to define the language, which is why JavaScript will run in *every* web browser regardless of the operating system or device.

## What You Need to Get Started with JavaScript and jQuery

- 1 Web Browser
- 2 Text Editor
- 3 Developer Tools (optional)

One of JavaScript's greatest strengths is its simplicity. It can be written and run on any operating system, and the only requirements are a web browser and a text editor. There are also numerous tools that can make JavaScript development more productive, but they are completely optional.

### Developer Tools

Commonly referred to as "developer tools," many browsers ship with built-in features that provide better insight into JavaScript and jQuery while they run in the browser. Although they aren't required, you may find developer tools helpful when it comes to debugging errors in your code. Check out these browsers' developer tools:

- [Apple Safari](#)
- [Google Chrome Developer Tools](#)
- [Microsoft Internet Explorer](#)
- [Mozilla Firefox Web Development Tools](#)
- [Opera Dragonfly](#)

# Running Code

## External

The first and recommended option is to write code in an external file (with a ".js" extension), which can then be included on our web page using an HTML `<script>` tag and pointing the `src` attribute to the file's location. Having JavaScript in a separate file will reduce code duplication if you want to reuse it on other pages. It will also allow the browser to cache the file on the remote client's computer, decreasing page load time.

```
1 | <!--Code is written in a .js file, then inclu  
2 | <script src="/path/to/example.js"></script>
```

## Inline

The second option is to inline the code directly on the web page. This is also achieved using HTML `<script>` tags, but instead of pointing the `src` attribute to a file, the code is placed between the tags. While there are use cases for this option, the majority of the time it is best to keep our code in an external file as described above.

```
1 | <!--Embed code directly on a web page using s  
2 | <script type="text/javascript">  
3 |     alert("Hello World!");  
4 | </script>
```

## Attributes

The last option is to use the event handler attributes of HTML

elements. This method is strongly discouraged:

```
1 | <!--Inline code directly on HTML elements bei
2 | <a href="javascript:alert("Hello World!");">C
3 | <button onClick="alert("Good Bye World!");">C1
```

## Placement

Placement of the previous two options is important and can vary depending on the situation. If you are including JavaScript that doesn't access the elements on the page, you can safely place the script before the closing HTML `<head>` tag. However, if the code will interact with the elements on the page, you have to make sure those elements exist at the time the script is executed. This common pitfall can be seen in the example below. The script for finding the element with the ID "hello-world" will be executed before the element is defined in the document.

```
1 | <!--Attempting to access an element too earl
2 | <!doctype html>
3 | <html>
4 | <head>
5 | <script type="text/javascript">
6 |
7 |     var title = document.getElementById("hel
8 |     console.log( title );
9 |
10 | </script>
11 | </head>
12 | <body>
13 |     <h1 id="hello-world">Hello World</h1>
14 | </body>
15 | </html>
```

It is a common pattern to move scripts to the bottom of the page, prior to the closing HTML `<body>` tag. This will guarantee that elements are defined when the script is executed.

```
1  <!--Moving the script to the bottom of the p
2  <!doctype html>
3  <html>
4  <head>
5  </head>
6  <body>
7      <h1 id="hello-world">Hello World</h1>
8      <script type="text/javascript">
9
10         var title = document.getElementById("r
11         console.log( title );
12
13     </script>
14 </body>
15 </html>
```



# Syntax Basics

## Comments

JavaScript has support for single and multi-line comments. Comments are ignored by the JavaScript engine and therefore have no side-effects on the outcome of the program. Use comments to document the code for other developers. Libraries like [JSDoc](#) are available to help generate project documentation pages based on commenting conventions.

```
1 // Single and multi line comments.
2 // this is an example of a single line comment.
3
4 /*
5  * this is an example
6  * of a
7  * multi line
8  * comment.
9  */
```

## Whitespace

Whitespace is also ignored in JavaScript. There are many tools that will strip out all the whitespace in a program, reducing the overall file size and improving network latency. Given the availability of tools like these, whitespace should be leveraged to make the code as readable as possible.

```
1 // Whitespace is insignificant.
2 var hello = "Hello";
3
4 var world    =    "World!";
```

```
1 // Semantic whitespace promotes readability.
2 // Readable code is good!
3 var foo = function() {
4
5     for ( var i = 0; i < 10; i++ ) {
6
7         alert( i );
8
9     }
10
11 };
12
13 foo();
14
15 // This is much harder to read!
16 var foo=function() {for(var i=0;i<10;++){ale
```

## Reserved Words

There are a handful of reserved words that can't be used when declaring user-defined variables and functions. Some of these reserved words are currently implemented, some are saved for future use, and others are reserved for historical reasons. A list of words and in-depth explanations for each can be found on the [MDN JavaScript Reference](#) site.

## Identifiers

Identifiers are used to give variables and functions a unique name so they can subsequently be referred to by that name. The name of an identifier must follow a few rules:

- Cannot be a reserved word.
- Can only be composed of letters, numbers, dollar signs, and underscores.
- The first character cannot be a number.

It's a best practice to name identifiers in a way that will make sense to you and other developers later on.

```
1 // Valid identifier names.  
2 var myAwesomeVariable = "a";  
3 var myAwesomeVariable2 = "b";  
4 var my_awesome_variable = "c";  
5 var $my_AwesomeVariable = "d";  
6 var _my_awesome_variable_$ = "e";
```

# Types

Types in JavaScript fall into two categories: primitives or objects. Primitive types include:

- String
- Number
- Boolean
- Null
- Undefined

## String

Strings are text wrapped in single or double quotation marks. It is best practice to consistently use one or the other. There may be times when the string contains quotation marks that collide with the ones used to create the string. In this case, either escape the characters using a `\` backslash or use different quotes around the string.

```
1 // Strings can created with double or single
2 var a = "I am a string";
3 var b = 'So am I!';
4
5 alert( a );
6
7 alert( b );
```

```
1 // Sometimes a string may contain quotation m
2 var statement1 = 'He said "JavaScript is awes
3
4 var statement2 = "He said \"JavaScript is awe
```

## Number

Number types are any positive or negative numeric value. There is no distinction between integer and floating point values.

```
1 // Numbers are any whole or floating point ir
2 var num1 = 100;
3 var num2 = 100.10;
4 var num3 = 0.10;
```

## Boolean

Boolean types are either true or false.

```
1 // Boolean values.
2 var okay = true;
3 var fail = false;
```

## Null and Undefined

Null and undefined are special types in JavaScript. Null types are a value that represent the absence of a value, similar to many other programming languages. Undefined types represent a state in which no value has been assigned at all. This type is created in two ways: by using the undefined keyword or by not defining a value at all.

```
1 // Two ways to achieve an undefined value.
2 var foo = null;
3
4 var bar1 = undefined;
5 var bar2;
```



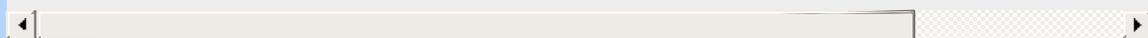
# Objects

Everything else in JavaScript is considered an Object. While there are [numerous built-in objects](#), this chapter will cover:

- Object
- Array
- Function

The simplest way to create an object is either through the Object constructor or the shorthand syntax known as object literal. These simple objects are unordered key/value pairs. The key is formally known as a property and the value can be any valid JavaScript type, even another object. To create or access a property on an object, we use what is known as "dot notation" or "bracket notation."

```
1 // Creating an object with the constructor:
2 var person1 = new Object;
3
4 person1.firstName = "John";
5 person1.lastName = "Doe";
6
7 alert( person1.firstName + " " + person1.las
8
9 // Creating an object with the object litera
10 var person2 = {
11     firstName: "Jane",
12     lastName: "Doe"
13 };
14
15 alert( person2.firstName + " " + person2.las
```



```
1 // As mentioned, objects can also have object
```

```
2 var people = {};  
3  
4 people["person1"] = person1;  
5 people["person2"] = person2;  
6  
7 alert( people["person1"].firstName );  
8 alert( people["person2"].firstName );
```

If a property is accessed that has not been defined, it will return a type of undefined.

```
1 // Properties that have not been created are  
2 var person = { name: "John Doe" };  
3  
4 alert( person.email ); // => undefined
```

Objects are covered further in the [Objects](#) section.

## Array

Arrays are a type of object that are ordered by the index of each item it contains. The index starts at zero and extends to however many items have been added, which is a property of the array known as the "length" of the array. Similar to a basic object, an array can be created with the array constructor or the shorthand syntax known as array literal.

```
1 // Creating an array with the constructor:  
2 var foo = new Array;  
3  
4 // Creating an array with the array literal s  
5 var bar = [];
```

There is an important distinction to be made between the two. Both an array constructor and an array literal can contain items to be added to the array upon creating it. However, if just a single numeric item is passed in, the array constructor will assume its length to be that value.

```
1 // The array literal returns a bar.length va
2 var foo = [ 100 ];
3
4 // => 100
5 alert( foo[0] );
6 // => 1
7 alert( foo.length );
8
9 // The array constructor returns a bar.lengt
10 var bar = new Array( 100 );
11 // => undefined
12 alert( bar[0] );
13 // => 100
14 alert( bar.length );
```

An array can be manipulated through methods that are available on the instance of the array. Items in the array can be accessed using bracket notation with a given. If the index does not exist or contains no value, the return type will be undefined.

A few common array methods are shown below:

```
1 // Using the push(), pop(), unshift() and sh
2 var foo = [];
3
4 foo.push("a");
5 foo.push("b");
6
7 alert( foo[ 0 ] ); // => a
8 alert( foo[ 1 ] ); // => b
```

```
9
10 alert( foo.length ); // => 2
11
12 foo.pop();
13
14 alert( foo[ 0 ] ); // => a
15 alert( foo[ 1 ] ); // => undefined
16
17 alert( foo.length ); // => 1
18
19 foo.unshift("z");
20
21 alert( foo[ 0 ] ); => z
22 alert( foo[ 1 ] ); => a
23
24 alert( foo.length ); => 2
25
26 foo.shift();
27
28 alert( foo[ 0 ] ); // => a
29 alert( foo[ 1 ] ); // => undefined
30
31 alert( foo.length ); // => 1
```

There are many more methods for manipulating arrays, some of which are covered further in the [Arrays](#) section. Details can be found on the [Mozilla Developer Network](#).

## Type Checking with jQuery

jQuery offers a few basic utility methods for determining the type of a specific value. Type checking is covered further in the [Testing Type](#) section, but here are some examples:

```
1 // Checking the type of an arbitrary value
2 var myValue = [ 1, 2, 3 ];
3
4 // Using JavaScript's typeof operator to tes
5
6 // false
7 typeof myValue === "string";
8 // false
9 typeof myValue === "number";
10 // false
11 typeof myValue === "undefined";
12 // false
13 typeof myValue === "boolean";
14
15 // Using strict equality operator to check f
16 // false
17 myValue === null;
18
19 // Using jQuery's methods to check for non-p
20
21 // false
22 jQuery.isFunction( myValue );
23 // false
24 jQuery.isPlainObject( myValue );
25 // true
26 jQuery.isArray( myValue );
```

# Operators

Basic operators allow you to manipulate values.

```
1 // Concatenation
2 var foo = "hello";
3 var bar = "world";
4
5 console.log( foo + " " + bar ); // "hello wor
```

```
1 // Multiplication and division
2 2 * 3;
3 2 / 3;
```

```
1 // Incrementing and decrementing
2 // The pre-increment operator increments the
3 // pre-increment:
4 var i = 1;
5
6 console.log( ++i ); // 2
7 console.log( i ); // 2
8
9 // The post-increment operator increments th
10 // post-increment:
11 var i = 1;
12
13 console.log( i++ ); // 1 - because i was 1
14 console.log( i ); // 2 - incremented after
```

## Operations on Numbers & Strings

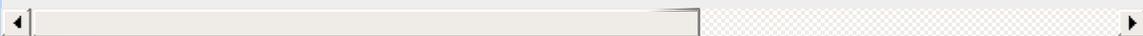
In JavaScript, numbers and strings will occasionally behave in unexpected ways.

```
1 // Addition vs. Concatenation
2 var foo = 1;
3 var bar = "2";
4
5 console.log( foo + bar ); // 12
```

```
1 // Coercing a string to act as a number:
2 var foo = 1;
3 var bar = "2";
4
5 console.log( foo + Number(bar) ); // 3
```

The Number constructor, when called as a function (as in the above example), will have the effect of casting its argument into a number. The unary plus operator also does the same thing:

```
1 // Forcing a string to act as a number (using
2 console.log( foo + +bar ); // 3
```



# Logical Operators

Logical operators allow evaluation of a series of operands using AND ( `&&` ) and OR ( `||` ) operations.

```
1 // Logical AND and OR operators
2 var foo = 1;
3 var bar = 0;
4 var baz = 2;
5
6 // returns 1, which is true
7 foo || bar;
8
9 // returns 1, which is true
10 bar || foo;
11
12 // returns 0, which is false
13 foo && bar;
14
15 // returns 2, which is true
16 foo && baz;
17
18 // returns 1, which is true
19 baz && foo;
```

In the above example, the `||` operator returns the value of the first truthy operand, or in cases where neither operand is truthy, it returns the last operand. The `&&` operator returns the value of the first false operand, or the value of the last operand if both operands are truthy.

You'll sometimes see developers use these logical operators for flow control instead of using `if` statements. For example:

```
1 // do something with foo if foo is truthy
2 foo && doSomething( foo );
```

```
3  
4 // set bar to baz if baz is truthy;  
5 // otherwise, set it to the return  
6 // value of createBar()  
7 var bar = baz || createBar();
```

This style is quite elegant and pleasantly terse; that said, it can be really hard to read or use, especially for beginners. See the section on truthy and falsy things in the [Conditional Code](#) article for more about evaluating truthiness.

## Comparison Operators

Comparison operators allow you to test whether values are equivalent or whether values are identical.

```
1 // Comparison operators
2 var foo = 1;
3 var bar = 0;
4 var baz = "1";
5 var bim = 2;
6
7 foo == bar; // false
8 foo != bar; // true
9 foo == baz; // true; but note that the typ
10
11 foo === baz; // false
12 foo !== baz; // true
13 foo === parseInt( baz ); // true
14
15 foo > bim; // false
16 bim > baz; // true
17 foo <= baz; // true
```

For more information about comparison operators, visit the [Mozilla Developer Network](#).

# Conditional Code

Sometimes a block of code should only be run under certain conditions. Flow control — via `if` and `else` blocks — lets you run code if certain conditions have been met.

```
1 // Flow control
2 var foo = true;
3 var bar = false;
4
5 if ( bar ) {
6     // this code will never run
7     console.log("hello!");
8 }
9
10 if ( bar ) {
11
12     // this code won't run
13
14 } else {
15
16     if ( foo ) {
17
18         // this code will run
19
20     } else {
21
22         // this code would run if foo and bar we
23
24     }
25
26 }
```

While curly braces aren't strictly required around single-line `if`

statements, using them consistently, even when they aren't strictly required, makes for vastly more readable code.

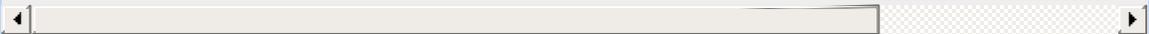
Be mindful not to define functions with the same name multiple times within separate if/else blocks, as doing so may not have the expected result.

## Truthy and Falsy Things

In order to use flow control successfully, it's important to understand which kinds of values are "truthy" and which kinds of values are "falsy." Sometimes, values that seem like they should evaluate one way actually evaluate another.

```
1 // Values that evaluate to true
2 "0";
3 "any string";
4 []; // an empty array
5 {}; // an empty object
6 1; // any non-zero number
```

```
1 // Values that evaluate to false
2 ""; // an empty string
3 NaN; // JavaScript's "not-a-number" variable
4 null;
5 undefined; // be careful -- undefined can be
```



## Conditional Variable Assignment with the Ternary Operator

Sometimes a variable should be set depending on some condition. An `if/else` statement works, but in many cases the ternary operator is more convenient. The ternary operator tests a condition; if the condition is true, it returns a certain value, otherwise it returns a different value.

The ternary operator:

```
1 | // set foo to 1 if bar is true;  
2 | // otherwise, set foo to 0  
3 | var foo = bar ? 1 : 0;
```

While the ternary operator can be used without assigning the return value to a variable, this is generally discouraged.

## Switch Statements

Rather than using a series of `if/else` blocks, sometimes it can be useful to use a `switch` statement instead. `Switch` statements look at the value of a variable or expression, and run different blocks of code depending on the value.

```
1 // A switch statement
2 switch ( foo ) {
3
4     case "bar":
5
6         alert("the value was bar -- yay!");
7
8     break;
9
10    case "baz":
11
12        alert("boo baz :(");
13
14    break;
15
16    default:
17
18        alert("everything else is just ok");
19
20 }
```

Switch statements have somewhat fallen out of favor in JavaScript, because often the same behavior can be accomplished by creating an object that has more potential for reuse or testing. For example:

```
1 var stuffToDo = {
2
3     "bar" : function() {
```

```
4
5     alert("the value was bar -- yay!");
6
7     },
8
9     "baz" : function() {
10
11         alert("boo baz :(");
12
13     },
14
15     "default" : function() {
16
17         alert("everything else is just ok");
18
19     }
20
21 };
22
23 if ( stuffToDo[ foo ] ) {
24
25     stuffToDo[ foo ]();
26
27 } else {
28
29     stuffToDo["default"]();
30
31 }
```

Objects are covered further in the [Types](#) and [Objects](#) sections.

# Loops

Loops let a block of code run a certain number of times:

```
1 // A for loop
2 // logs "try 0", "try 1", ..., "try 4"
3 for ( var i = 0; i < 5; i++ ) {
4
5     console.log( "try " + i );
6
7 }
```

Note that in loops, the variable `i` is not "scoped" to the loop block even though the keyword `var` is used before the variable name. Scope is covered in more depth in the [Scope](#) section.

## The `for` loop

A `for` loop is made up of four statements and has the following structure:

```
1  for ( [initialisation]; [conditional]; [itera
2
3    [ loopBody ]
4
5  }
```

The *initialisation* statement is executed only once, before the loop starts. It gives you an opportunity to prepare or declare any variables.

The *conditional* statement is executed before each iteration, and its return value decides whether the loop is to continue. If the conditional statement evaluates to a falsy value, then the loop stops.

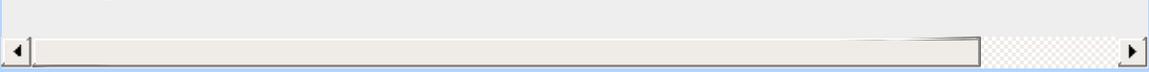
The *iteration* statement is executed at the end of each iteration and gives you an opportunity to change the state of important variables. Typically, this will involve incrementing or decrementing a counter and thus bringing the loop closer to its end.

The *loopBody* statement is what runs on every iteration. It can contain anything. Typically, there will be multiple statements that need to be executed, and should be wrapped in a block ( `{...}` ).

Here's a typical `for` loop:

```
1  // A typical for loop
2  for (var i = 0, limit = 100; i < limit; i++)
3
4    // This block will be executed 100 times
5
6    console.log( 'Currently at ' + i );
```

```
7  
8 // Note: the last log will be "Currently at 99"  
9  
10 }
```



## The `while` loop

A while loop is similar to an `if` statement, except that its body will keep executing until the condition evaluates to false.

```
1  while ( [conditional] ) {
2
3      [loopBody]
4
5  }
```

Here's a typical `while` loop:

```
1  // A typical while loop
2  var i = 0;
3
4  while ( i < 100 ) {
5
6      // This block will be executed 100 times
7      console.log( "Currently at " + i );
8
9      // increment i
10     i++;
11
12 }
```

Notice that the counter is incrementing within the loop's body. It's possible to combine the conditional and incrementer, like so:

```
1  // A while loop with a combined conditional a
2  var i = -1;
3
4  while ( ++i < 100 ) {
```

```
5  
6 // This block will be executed 100 times  
7 console.log( "Currently at " + i );  
8  
9 }
```



Notice that the counter starts at -1 and uses the prefix incrementer (++i).

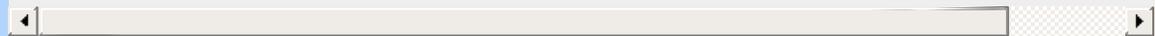
## The `do-while` loop

This is almost exactly the same as the `while` loop, except for the fact that the loop's body is executed at least once before the condition is tested.

```
1  do {  
2  
3    [ loopBody ]  
4  
5  } while ( [conditional] )
```

Here's a `do-while` loop:

```
1  // A do-while loop  
2  do {  
3  
4    // Even though the condition evaluates to f  
5    // this loop's body will still execute once  
6    alert("Hi there!");  
7  
8  } while ( false );
```



These types of loops are quite rare since only few situations require a loop that blindly executes at least once. Regardless, it's good to be aware of it.

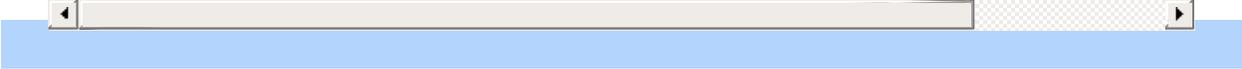
## Breaking and continuing

Usually, a loop's termination will result from the conditional statement not evaluating to true, but it is possible to stop a loop in its tracks from within the loop's body with the break statement.

```
1 // Stopping a loop
2 for ( var i = 0; i < 10; i++ ) {
3
4     if ( something ) {
5
6         break;
7
8     }
9
10 }
```

You may also want to continue the loop without executing more of the loop's body. This is done using the continue statement.

```
1 // Skipping to the next iteration of a loop
2 for ( var i = 0; i < 10; i++ ) {
3
4     if ( something ) {
5
6         continue;
7
8     }
9
10 // The following statement will only be ex
11 // if the conditional 'something' has not
12 console.log("I have been reached");
13
14 }
```



# Reserved Words

JavaScript has a number of “reserved words,” or words that have special meaning in the language. You should avoid using these words in your code except when using them with their intended meaning.

- abstract
- boolean
- break
- byte
- case
- catch
- char
- class
- const
- continue
- debugger
- default
- delete
- do
- double
- else
- enum
- export
- extends
- final
- finally
- float
- for

- function
- goto
- if
- implements
- import
- in
- instanceof
- int
- interface
- long
- native
- new
- package
- private
- protected
- public
- return
- short
- static
- super
- switch
- synchronized
- this
- throw
- throws
- transient
- try
- typeof
- var
- void

- volatile
- while
- with

# Arrays

Arrays are zero-indexed, ordered lists of values. They are a handy way to store a set of related items of the same type (such as strings), though in reality, an array can include multiple types of items, including other arrays.

To create an array, either use the object constructor or the literal declaration, by assigning the variable a list of values after the declaration.

```
1 // A simple array with constructor
2 var myArray1 = new Array( "hello", "world" );
3 // literal declaration, the preferred way
4 var myArray2 = [ "hello", "world" ];
```

The literal declaration is generally preferred. See the [Google Coding Guidelines](#) for more information.

If the values are unknown, it is also possible to declare an empty Array, and add elements either through functions or through accessing by index:

```
1 // Creating empty arrays and adding values
2 var myArray = [];
3
4 // adds "hello" on index 0
5 myArray.push("hello");
6
7 // adds "world" on index 1
8 myArray.push("world");
9
10 // adds "!" on index 2
11 myArray[ 2 ] = "!";
```

'push' is a function that adds an element on the end of the array and expands the array respectively. You also can directly add items by index. Missing indices will be filled with 'undefined'.

```
1 // Leaving indices
2 var myArray = [];
3
4 myArray[ 0 ] = "hello";
5 myArray[ 1 ] = "world";
6 myArray[ 3 ] = "!";
7
8 console.log( myArray ); // [ "hello", "world"
```

If the size of the array is unknown, 'push' is far more safe. You can both access and assign values to array items with the index.

```
1 // Accessing array items by index
2 var myArray = [ "hello", "world", "!" ];
3
4 console.log( myArray[2] ); // "!"
```

## Array Methods and Properties

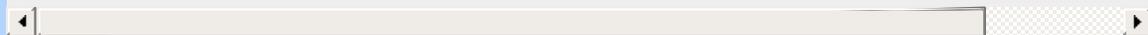
### `.length`

The `.length` property is used to determine the amount of items in an array.

```
1 // Length of an array
2 var myArray = [ "hello", "world", "!" ];
3
4 console.log( myArray.length ); // 3
```

You will need the `.length` property for looping through an array:

```
1 // For loops and arrays - a classic
2 var myArray = [ "hello", "world", "!" ];
3
4 for ( var i = 0; i < myArray.length; i = i +
5     console.log( myArray[i] );
6 }
```



Except when using `for/in` loops:

```
1 // For loops and arrays - alternate method
2 var myArray = [ "hello", "world", "!" ];
3
4 for ( var i in myArray ) {
5     console.log( myArray[ i ] );
6 }
```

### `.concat`

Concatenate two or more arrays with `.concat`:

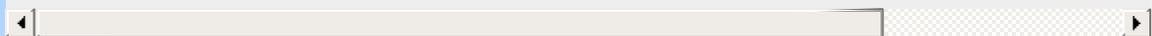
```
1 // Concatenating Arrays
2 var myArray = [ 2, 3, 4 ];
3 var myOtherArray = [ 5, 6, 7 ];
4
5 // [ 2, 3, 4, 5, 6, 7 ]
6 var wholeArray = myArray.concat( myOtherArray
```



`.join`

`.join` creates a string representation of the array. Its parameter is a string that works as a separator between elements (default separator is a comma):

```
1 // Joining elements
2 var myArray = [ "hello", "world", "!" ];
3
4 console.log( myArray.join(" ") ); // "hello
5 console.log( myArray.join() ); // "hello,
6 console.log( myArray.join("") ); // "hellow
7 console.log( myArray.join("!!") ); // "hello!
```



`.pop`

`.pop` removes the last element of an array. It is the opposite method of `.push`:

```
1 // pushing and popping
2 var myArray = [];
3
4 myArray.push( 0 ); // [ 0 ]
5 myArray.push( 2 ); // [ 0, 2 ]
6 myArray.push( 7 ); // [ 0, 2, 7 ]
7 myArray.pop(); // [ 0, 2 ]
```

### .reverse

As the name suggests, the elements of the array are in reverse order after calling this method:

```
1 // reverse
2 var myArray = [ "world" , "hello" ];
3
4 // [ "hello", "world" ]
5 myArray.reverse();
```

### .shift

Removes the first element of an array. With `.push` and `.shift`, you can recreate the method of a [queue](#):

```
1 // queue with shift() and push()
2 var myArray = [];
3
4 myArray.push( 0 ); // [ 0 ]
5 myArray.push( 2 ); // [ 0 , 2 ]
6 myArray.push( 7 ); // [ 0 , 2 , 7 ]
7 myArray.shift(); // [ 2 , 7 ]
```

### .slice

Extracts a part of the array and returns that part in a new array. This method takes one parameter, which is the starting index:

```
1 // slicing
2 var myArray = [ 1, 2, 3, 4, 5, 6, 7, 8 ];
3 var newArray = myArray.slice( 3 );
4
5 console.log( myArray ); // [ 1, 2, 3, 4, 5,
6 console.log( newArray ); // [ 4, 5, 6, 7, 8 ]
```

## `.splice`

Removes a certain amount of elements and adds new ones at the given index. It takes at least 3 parameters:

```
1 // splice method
2 myArray.splice( index, length, values, ... );
```

- *Index* - The starting index.
- *Length* - The number of elements to remove.
- *Values* - The values to be inserted at the Index position.

For example:

```
1 // splice example
2 var myArray = [ 0, 7, 8, 5 ];
3 myArray.splice( 1, 2, 1, 2, 3, 4 );
4
5 console.log( myArray ); // [ 0, 1, 2, 3, 4, 5 ]
```

## `.sort`

Sorts an array. It takes one parameter, which is a comparing function. If this function is not given, the array is sorted ascending:

```
1 // sorting without comparing function
2 var myArray = [ 3, 4, 6, 1 ];
3
4 myArray.sort(); // 1, 3, 4, 6
```

```
1 // sorting with comparing function
```

```
2 function descending( a, b ) {
3   return b - a;
4 }
5
6 var myArray = [ 3, 4, 6, 1 ];
7
8 myArray.sort( descending ); // [ 6, 4, 3, 1 ]
```

The return value of descending (for this example) is important. If the return value is less than zero, the index of a is before b, and if it is greater than zero it's vice-versa. If the return value is zero, the elements index is equal.

`.unshift`

Inserts an element at the first position of the array:

```
1 // unshift
2 var myArray = [];
3
4 myArray.unshift( 0 ); // [ 0 ]
5 myArray.unshift( 2 ); // [ 2, 0 ]
6 myArray.unshift( 7 ); // [ 7, 2, 0 ]
```

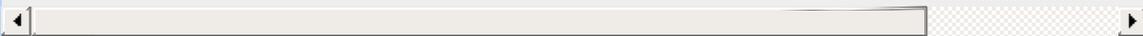
`.forEach`

In modern browsers it is possible to traverse through arrays with a `.forEach` method, where you pass a function that is called for each element in the array.

The function takes up to three arguments: *Element* - The element itself. *Index* - The index of this element in the array. *Array\** - The array itself.

All of these are optional, but you will need at least the 'element' parameter in most cases.

```
1 // native forEach
2 function printElement( elem ) {
3     console.log( elem );
4 }
5
6 function printElementAndIndex( elem, index ) {
7     console.log( "Index " + index + ": " + elem );
8 }
9
10 function negateElement( elem, index, array ) {
11     array[ index ] = -elem;
12 }
13
14 myArray = [ 1, 2, 3, 4, 5 ];
15
16 // prints all elements to the console
17 myArray.forEach( printElement );
18
19 // prints "Index 0: 1" "Index 1: 2" "Index 2: 3" ...
20 myArray.forEach( printElementAndIndex );
21
22 // myArray is now [ -1, -2, -3, -4, -5 ]
23 myArray.forEach( negateElement );
```



# Objects

Objects contain one or more key-value pairs. The key portion can be any string. The value portion can be any type of value: a number, a string, an array, a function, or even another object. When one of these values is a function, it's called a method of the object. Otherwise, they are called properties.

As it turns out, nearly everything in JavaScript is an object — arrays, functions, numbers, even strings — and they all have properties and methods.

```
1 // Creating an object literal
2 var myObject = {
3   sayHello : function() {
4     console.log("hello");
5   },
6   myName : "Rebecca"
7 };
8
9 myObject.sayHello(); // "hello"
10
11 console.log( myObject.myName ); // "Rebecca"
```

When creating object literals, note that the key portion of each key-value pair can be written as any valid JavaScript identifier, a string (wrapped in quotes), or a number:

```
1 // test
2 var myObject = {
3   validIdentifier: 123,
4   "some string": 456,
5   99999: 789
6 };
```



# Functions

Functions contain blocks of code that need to be executed repeatedly. Functions can take zero or more arguments, and can optionally return a value.

Functions can be created in a variety of ways, two of which are shown below:

```
1 // Function Declaration
2 function foo() {
3
4     /* do something */
5
6 }
```

```
1 // Named Function Expression
2 var foo = function() {
3
4     /* do something */
5
6 }
```

## Using Functions

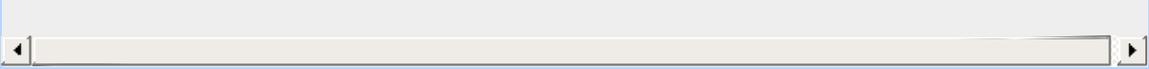
```
1 // A simple function
2 var greet = function( person, greeting ) {
3
4     var text = greeting + ", " + person;
5
6     console.log( text );
7
8 };
9
10 greet( "Rebecca", "Hello" );
```

```
1 // A function that returns a value
2 var greet = function( person, greeting ) {
3
4     var text = greeting + ", " + person;
5
6     return text;
7
8 };
9
10 console.log( greet( "Rebecca", "hello" ) );
```



```
1 // A function that returns another function
2 var greet = function( person, greeting ) {
3     var text = greeting + ", " + person;
4
5     return function() {
6         console.log( text );
7     };
8 }
```

```
8   };  
9  
10  var greeting = greet( "Rebecca", "Hello" );  
11  
12  greeting();
```



## Immediately-Invoked Function Expression (IIFE)

A common pattern in JavaScript is the immediately-invoked function expression. This pattern creates a function expression and then immediately executes the function. This pattern is extremely useful for cases where you want to avoid polluting the global namespace with code — no variables declared inside of the function are visible outside of it.

```
1 // An immediately-invoked function expression
2 (function() {
3
4     var foo = "Hello world";
5
6 })();
7
8 console.log( foo ); // undefined!
```

## Functions as Arguments

In JavaScript, functions are "first-class citizens" — they can be assigned to variables or passed to other functions as arguments. Passing functions as arguments is an extremely common idiom in jQuery.

```
1 // Passing an anonymous function as an argum
2 var myFn = function( fn ) {
3     var result = fn();
4     console.log( result );
5 };
6
7 // logs "hello world"
8 myFn( function() {
9     return "hello world";
10 });
```

```
1 // Passing a named function as an argument
2
3 var myFn = function( fn ) {
4     var result = fn();
5     console.log( result );
6 };
7
8 var myOtherFn = function() {
9     return "hello world";
10 };
11
12 myFn( myOtherFn ); // "hello world"
```

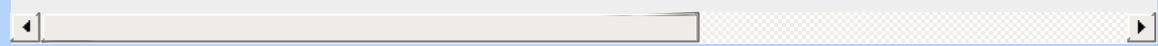
# Testing Type

JavaScript offers a way to test the type of a variable. However, the result can be confusing — for example, the type of an array is "Object."

It's common practice to use the `typeof` operator when trying to determine the type of a specific value.

```
1 // Testing the type of various variables
2 var myFunction = function() {
3   console.log("hello");
4 };
5 var myObject = {
6   foo : "bar"
7 };
8 var myArray = [ "a", "b", "c" ];
9 var myString = "hello";
10 var myNumber = 3;
11
12 typeof myFunction; // "function"
13 typeof myObject; // "object"
14 typeof myArray; // "object" -- Careful!
15 typeof myString; // "string"
16 typeof myNumber; // "number"
17 typeof null; // "object" -- Careful!
18
19 if ( myArray.push && myArray.slice && myArra
20
21 // probably an array (this is called "duck
22 }
23
24 if ( Object.prototype.toString.call( myArray
25
26 // Definitely an array!
27 // This is widely considered as the most r
```

```
28 | // to determine if a specific value is an Array.  
29 | }
```



jQuery also offers utility methods to help determine the type of an arbitrary value.

# The “this” Keyword

In JavaScript, as in most object-oriented programming languages, `this` is a special keyword that is used in methods to refer to the object on which a method is being invoked. The value of `this` is determined using a simple series of steps:

- If the function is invoked using `Function.call` or `Function.apply`, `this` will be set to the first argument passed to `call/apply`. If the first argument passed to `call/apply` is null or undefined, `this` will refer to the global object (which is the `window` object in web browsers).
- If the function being invoked was created using `Function.bind`, `this` will be the first argument that was passed to `bind` at the time the function was created.
- If the function is being invoked as a method of an object, `this` will refer to that object.
- Otherwise, the function is being invoked as a standalone function not attached to any object, and `this` will refer to the global object.

```
1 // A function invoked using Function.call
2 var myObject = {
3   sayHello: function() {
4     console.log( "Hi! My name is " + this.my
5   },
6   myName: "Rebecca"
7 };
8
9 var secondObject = {
10  myName: "Colin"
11 };
12
13 myObject.sayHello(); // '
14 myObject.sayHello.call( secondObject ); // '

```

```
1 // A function created using Function.bind
2 var myName = "the global object";
3 var sayHello = function() {
4     console.log( "Hi! My name is " + this.myName);
5 };
6 var myObject = {
7     myName: "Rebecca"
8 };
9 var myObjectHello = sayHello.bind( myObject
10
11 sayHello()); // "Hi! My name is the global object"
12 myObjectHello(); // "Hi! My name is Rebecca"
```

```
1 // A function being attached to an object at runtime
2 var myName = "the global object";
3 var sayHello = function() {
4     console.log( "Hi! My name is " + this.myName);
5 };
6 var myObject = {
7     myName: "Rebecca"
8 };
9 var secondObject = {
10     myName: "Colin"
11 };
12
13 myObject.sayHello = sayHello;
14 secondObject.sayHello = sayHello;
15
16 sayHello(); // "Hi! My name is the global object"
17 myObject.sayHello(); // "Hi! My name is Rebecca"
18 secondObject.sayHello(); // "Hi! My name is Colin"
```

When invoking a function deep within a long namespace, it is often tempting to reduce the amount of code you need to type by storing a reference to the actual function as a single, shorter variable. It is important not to do this with instance methods as this will cause the value of `this` within the function to change, leading to incorrect code operation. For instance:

```
1  var myNamespace = {
2    myObject: {
3      sayHello: function() {
4        console.log( "Hi! My name is " + this.
5          },
6        myName: "Rebecca"
7      }
8    };
9
10  var hello = myNamespace.myObject.sayHello;
11
12  hello(); // "Hi! My name is undefined"
```

You can, however, safely reduce everything up to the object on which the method is invoked:

```
1  var myNamespace = {
2    myObject: {
3      sayHello: function() {
4        console.log( "Hi! My name is " + this.
5          },
6        myName: "Rebecca"
7      }
8    };
9
10  var obj = myNamespace.myObject;
11
12  obj.sayHello(); // "Hi! My name is Rebecca"
```



# Scope

"Scope" refers to the variables that are available to a piece of code at a given time. A lack of understanding of scope can lead to frustrating debugging experiences.

When a variable is declared inside of a function using the `var` keyword, it is only available to code inside of that function — code outside of that function cannot access the variable. On the other hand, functions defined inside that function will have access to the declared variable.

Furthermore, variables that are declared inside a function without the `var` keyword are not local to the function — JavaScript will traverse the scope chain all the way up to the window scope to find where the variable was previously defined. If the variable wasn't previously defined, it will be defined in the global scope, which can have unexpected consequences.

```
1 // Functions have access to variables defined
2 var foo = "hello";
3 var sayHello = function() {
4     console.log( foo );
5 };
6
7 sayHello(); // "hello"
8 console.log( foo ); // "hello"
```

```
1 // Code outside the scope in which a variable
2 var sayHello = function() {
3     var foo = "hello";
4     console.log( foo );
5 };
6
7 sayHello(); // hello
```

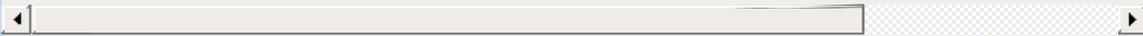
```
8
9 console.log( foo ); // undefined
```

```
1 // Variables with the same name can exist in
2 var foo = "world";
3
4 var sayHello = function() {
5     var foo = "hello";
6     console.log( foo );
7 };
8
9 sayHello(); // logs "hello"
10 console.log( foo ); // logs "world"
```

```
1 // Functions can see changes in variable val
2 var myFunction = function() {
3     var foo = "hello";
4     var myFn = function() {
5         console.log( foo );
6     };
7     foo = "world";
8     return myFn;
9 };
10
11 var f = myFunction();
12 f(); // "world"
```

```
1 // Scope insanity
2 // a self-executing anonymous function
3 (function() {
4     var baz = 1;
```

```
5     var bim = function() {
6         alert( baz );
7     };
8
9     bar = function() {
10        alert( baz );
11    };
12
13    })();
14
15    // baz is not defined outside of the function
16    console.log( baz );
17
18    // bar is defined outside of the anonymous function
19    // because it wasn't declared with var; furthermore,
20    // because it was defined in the same scope as baz,
21    // it has access to baz even though other code
22    // outside of the function does not
23    bar();
24
25    // bim is not defined outside of the anonymous function
26    // so this will result in an error
27    bim();
```



# Closures

Closures are an extension of the concept of scope. With closures, functions have access to variables that were available in the scope where the function was created. If that seems confusing, don't worry: closures are generally best understood by example.

As shown in the [Functions](#) section, functions have access to changing variable values. The same sort of behavior exists with functions defined within loops — the function "sees" the change in the variable's value even after the function is defined, resulting in each function referencing the last value stored in the variable.

```
1 // Each function executed within the loop wi
2 // the last value stored in i (5).
3 // this won't behave as we want it to -
4 // every 100 milliseconds, 5 will alert
5 for ( var i = 0; i < 5; i++ ) {
6
7     setTimeout(function() {
8
9         alert( i );
10
11     }, i * 100 );
12
13 }
```

Closures can be used to prevent this by creating a unique scope for each iteration — storing each unique value of the variable within its scope.

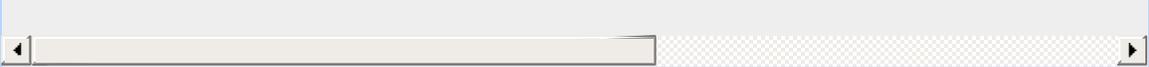
```
1 // Using a closure to create a new private s
2 // fix: "close" the value of i inside create
3 var createFunction = function( i ) {
```

```
4
5     return function() {
6
7         alert( i );
8
9     };
10
11 };
12
13 for ( var i = 0; i < 5; i++ ) {
14
15     setTimeout( createFunction( i ), i * 100 );
16
17 }
```

Closures can also be used to resolve issues with the `this` keyword, which is unique to each scope:

```
1 //Using a closure to access inner and outer
2 var outerObj = {
3     myName : "outer",
4     outerFunction : function() {
5
6         // provide a reference to outerObj throu
7         var self = this;
8
9         var innerObj = {
10            myName : "inner",
11            innerFunction : function() {
12
13                console.log( self.myName, this.myName
14
15            }
16        };
17 }
```

```
18     innerObj.innerFunction();
19
20     console.log( this.myName ); // "outer"
21 }
22 };
23
24 outerObj.outerFunction();
```



## Function.bind

Closures can be particularly useful when dealing with callbacks. However, it is often better to use `Function.bind`, which will avoid any overhead associated with scope traversal.

`Function.bind` is used to create a new function. When called, the new function then calls itself in the context of the supplied `this` value, using a given set of arguments that will precede any arguments provided when the new function was initially called.

As `bind` is a recent addition to ECMAScript 5, it may not be present in all browsers, which is something to be wary of when deciding whether to use it. However, it's possible to work around support by using [this shim](#) from MDN:

```
1 // Shim from MDN
2 if (!Function.prototype.bind) {
3
4     Function.prototype.bind = function( oThis
5
6         if (typeof this !== "function") {
7
8             // closest thing possible to the ECMAS
9             // IsCallable function
10            throw new TypeError("Function.prototype
11
12        }
13
14        var fSlice = Array.prototype.slice,
15            aArgs = fSlice.call( arguments, 1 ),
16            fToBind = this,
17            fNOP = function() {},
18            fBound = function() {
19
20                return fToBind.apply( this instanc
21                                    ? this
22                                    : oThis ||
```

```
23         aArgs.concat( fSlice.ca
24
25         };
26
27         fNOP.prototype = this.prototype;
28         fBound.prototype = new fNOP();
29
30         return fBound;
31
32     };
33
34 }
```

One of the simplest uses of `bind` is making a function that is called with a particular value for `this`, regardless of how it's called. A common mistake developers make is attempting to extract a method from an object, then later calling that method and expecting it to use the origin object as its `this`. However, this can be solved by creating a bound function using the original object as demonstrated below:

```
1  //let's manipulate "this" with a basic example
2  var user = "johnsmith";
3  var module = {
4
5      getUser: function() {
6
7          return this.user;
8
9      },
10
11     user: "janedoe"
12
13 };
14
15 // module.getUser() is called where "module"
```

```
16 // and "module.user" is returned.
17
18 // janedoe
19 module.getUser();
20
21 // let's now store a reference in the global version of
22 var getUser = module.getUser;
23
24 // getUser() called, "this" is global, "user" is returned
25
26 // johnsmith
27 getUser();
28
29 // store a ref with "module" bound as "this"
30 var boundGetUser = getUser.bind( module );
31
32 // boundGetUser() called, "module" is "this" again, "module"
33
34 // janedoe
35 boundGetUser();
```



# Using jQuery Core

- [\\$ vs \\$\(\)](#)
- [\\$\( document \).ready\(\)](#)
- [Avoiding Conflicts with Other Libraries](#)
- [Attributes](#)
- [Selecting Elements](#)
- [Working with Selections](#)
- [Manipulating Elements](#)
- [The jQuery Object](#)
- [Traversing](#)
- [CSS, Styling, & Dimensions](#)
- [Data Methods](#)
- [Utility Methods](#)
- [Iterating over jQuery and non-jQuery Objects](#)
- [Using jQuery's .index\(\) Function](#)
- [Frequently Asked Questions](#)
  - [How do I select an item using class or ID?](#)
  - [How do I select elements when I already have a DOM element?](#)
  - [How do I test whether an element has a particular class?](#)
  - [How do I test whether an element exists?](#)
  - [How do I determine the state of a toggled element?](#)
  - [How do I select an element by an ID that has characters used in CSS notation?](#)
  - [How do I disable/enable a form element?](#)
  - [How do I check/uncheck a checkbox input or radio button?](#)
  - [How do I get the text value of a selected option?](#)
  - [How do I replace text from the 3rd element of a list of 10 items?](#)

- [How do I pull a native DOM element from a jQuery object?](#)

## \$ vs \$()

Until now, we've been dealing entirely with methods that are called on a jQuery object. For example:

```
1 | $("h1").remove();
```

Most jQuery methods are called on jQuery objects as shown above; these methods are said to be part of the `$.fn` namespace, or the “jQuery prototype,” and are best thought of as jQuery object methods.

However, there are several methods that do not act on a selection; these methods are said to be part of the jQuery namespace, and are best thought of as core jQuery methods.

This distinction can be incredibly confusing to new jQuery users. Here's what you need to remember:

- Methods called on jQuery selections are in the `$.fn` namespace, and automatically receive and return the selection as this.
- Methods in the `$` namespace are generally utility-type methods, and do not work with selections; they are not automatically passed any arguments, and their return value will vary.

There are a few cases where object methods and core methods have the same names, such as `$.each` and `$.fn.each`. In these cases, be extremely careful when reading the documentation that you are exploring the correct method.

# `$( document ).ready()`

A page can't be manipulated safely until the document is “ready.” jQuery detects this state of readiness for you. Code included inside `$( document ).ready()` will only run once the page DOM (Document Object Model) is ready for JavaScript code to execute. Code included inside `$( window ).load(function(){ ... })` will run once the entire page (images or iframes), not just the DOM, is ready.

```
1 // A $( document ).ready() block
2 $( document ).ready(function() {
3     console.log("ready!");
4 });
```

Experienced developers sometimes use shorthand for `$( document ).ready()`. If you are writing code that people who aren't experienced with jQuery may see, it's best to use the long form.

```
1 // Shorthand for $( document ).ready()
2 $(function() {
3     console.log("ready!");
4 });
```

You can also pass a named function to `$( document ).ready()` instead of passing an anonymous function.

```
1 // Passing a named function instead of an anc
2
3 function readyFn( jQuery ) {
4     // code to run when the document is ready
5 }
6
7 $( document ).ready( readyFn );
```

```
8 // OR
9 $( window ).load( readyFn );
```

The below example shows `$( document ).ready()` and `$( window ).load()` in action. The code tries to load a website url in an iframe and checks for both events:

```
1 <html>
2   <head>
3     <script src="http://code.jquery.com/jque
4     <script>
5       $( document ).ready(function() {
6         console.log("document loaded");
7       });
8
9       $( window ).load(function() {
10        console.log("window loaded");
11      });
12    </script>
13  </head>
14  <body>
15    <iframe src="http://techcrunch.com"></
16    </body>
17 </html>
```

# Avoiding Conflicts with Other Libraries

The jQuery library and virtually all of its plugins are contained within the `jQuery` namespace. As a general rule, global objects are stored inside the jQuery namespace as well, so you shouldn't get a clash between jQuery and any other library (like prototype.js, MooTools, or YUI).

That said, there is one caveat: *by default, jQuery uses `$` as a shortcut for `jQuery`*. Thus, if you are using another JavaScript library that uses the `$` variable, you can run into conflicts with jQuery. In order to avoid these conflicts, you need to put jQuery in no-conflict mode immediately after it is loaded onto the page and before you attempt to use jQuery in your page.

## Putting jQuery into No-Conflict Mode

When you put jQuery into no-conflict mode, you have the option of assigning a new variable name to replace the `$` alias.

```
1  <!-- Putting jQuery into no-conflict mode -->
2  <script src="prototype.js"></script>
3  <script src="jquery.js"></script>
4  <script>
5      // $j is now an alias to the jQuery func
6      // creating the new alias is optional
7      var $j = jQuery.noConflict();
8      $j(document).ready(function(){
9          $j("div").hide();
10     });
11
12     // The $ variable now has the prototype
13     // which is a shortcut for document.getE
14     // mainDiv below is a DOM element, not a
15     window.onload = function(){
16         var mainDiv = $('main');
17     }
18 </script>
```

In the code above, the `$` will revert back to its meaning in original library. You'll still be able to use the full function name `jQuery` as well as the new alias `$j` in the rest of your application. The new alias can be named anything you'd like: `jq`, `$J`, `awesomeQuery`, etc.

Finally, if you don't want to define another alternative to the full `jQuery` function name (you really like to use `$` and don't care about using the other library's `$` method), then there's still another approach you might try: simply add the `$` as an argument passed to your `jQuery(document).ready()` function. This is most frequently used in the case where you still want the benefits of really concise jQuery code, but don't want to cause conflicts with other libraries.

```
1  <!-- Another way to put jQuery into no-confli
2  <script src="prototype.js"></script>
3  <script src="jquery.js"></script>
4  <script>
5
6      jQuery.noConflict();
7      jQuery(document).ready(function($){
8          // You can use the locally-scoped $ i
9          $("div").hide();
10     });
11
12     // The $ variable in the global scope ha
13     window.onload = function(){
14         var mainDiv = $('main');
15     }
16
17 </script>
```

This is probably the ideal solution for most of your code, considering that there'll be less code that you'll have to change in order to achieve complete compatibility.

## Including jQuery Before Other Libraries

The code snippets above rely on jQuery being loaded after prototype.js is loaded. If you include jQuery before other libraries, you may use `jQuery` when you do some work with jQuery, but the `$` will have the meaning defined in the other library. There is no need to relinquish the `$` alias by calling `jQuery.noConflict()`.

```
1  <!-- Loading jQuery before other libraries -
2  <script src="jquery.js"></script>
3  <script src="prototype.js"></script>
4  <script>
5
6      // Use full jQuery function name to refer
7      jQuery(document).ready(function(){
8          jQuery("div").hide();
9      });
10
11     // Use the $ variable as defined in prot
12     window.onload = function() {
13         var mainDiv = $('main');
14     };
15
16 </script>
```



## Summary of Ways to Reference the jQuery Function

Here's a recap of ways you can reference the jQuery function when the presence of another library creates a conflict over the use of the `$` variable:

### Create a New Alias

The `jQuery.noConflict()` method returns a reference to the jQuery function, so you can capture it in whatever variable you'd like:

```
1 <script src="prototype.js"></script>
2 <script src="jquery.js"></script>
3 <script>
4
5     // Give $ back to prototype.js; create ne
6     var $jq = jQuery.noConflict();
7
8 </script>
```

### Use an Immediately Invoked Function Expression

You can continue to use the standard `$` by wrapping your code in an immediately invoked function expression; this is also a standard pattern for jQuery plugin authoring, where the author cannot know whether another library will have taken over the `$`. See the [Plugins](#) section for more information about writing plugins.

```
1 <!-- Using the $ inside an immediately-invok
2 <script src="prototype.js"></script>
3 <script src="jquery.js"></script>
4 <script>
```

```
5     jQuery.noConflict();
6
7     (function($) {
8         // your jQuery code here, using the $
9     })(jQuery);
10 </script>
```

Note that if you use this technique, you will not be able to use prototype.js methods inside the immediately invoked function that expect `$` to be prototype.js's `$`.

## Use the Argument That's Passed to the `jQuery(document).ready()` Function

```
1 <script src="jquery.js"></script>
2 <script src="prototype.js"></script>
3 <script>
4
5     jQuery(document).ready(function($){
6         // your jQuery code here, using $ to re
7     });
8
9 </script>
```

Or using the more concise syntax for the DOM ready function:

```
1 <script src="jquery.js"></script>
2 <script src="prototype.js"></script>
3 <script>
4
5     jQuery(function($){
6         // your jQuery code here, using the $
7     });
```

8

9 `</script>`

# Attributes

An element's attributes can contain useful information for your application, so it's important to be able to get and set them.

`$.fn.attr`

The `$.fn.attr` method acts as both a getter and a setter. As a setter, `$.fn.attr` can accept either a key and a value, or an object containing one or more key/value pairs.

`$.fn.attr` as a setter:

```
1 // Setting attributes
2 $("a").attr( "href", "allMyHrefsAreTheSameNow" );
3
4 $("a").attr({
5   title: "all titles are the same too!",
6   href: "somethingNew.html"
7 });
```

`$.fn.attr` as a getter:

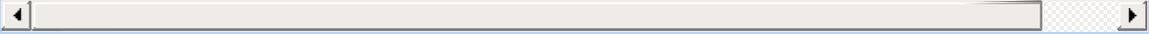
```
1 // Getting attributes
2 $("a").attr("href"); // returns the href for
```

# Selecting Elements

The most basic concept of jQuery is to “select some elements and do something with them.” jQuery supports most CSS3 selectors, as well as some non-standard selectors. For a complete selector reference, visit the [Selectors documentation on api.jquery.com](http://api.jquery.com/selectors).

## Selecting Elements by ID

```
1 // Selecting elements by ID  
2 $("#myId"); // note IDs must be unique per page
```



## Selecting Elements by Class Name

```
1 | // Selecting elements by class name  
2 | $(".myClass");
```

## Selecting Elements by Attribute

```
1 // Selecting elements by attribute  
2 $("input[name='first_name']"); // beware, thi
```



## Selecting Elements by Compound CSS Selector

```
1 // Selecting elements by compound CSS selector
2 $("#contents ul.people li");
```

## Pseudo-selectors

```
1 // Pseudo-selectors
2 $("a.external:first");
3 $("tr:odd");
4
5 // select all input-like elements in a form
6 $("#myForm :input");
7 $("div:visible");
8
9 // all except the first three divs
10 $("div:gt(2)");
11
12 // all currently animated divs
13 $("div:animated");
```

**Note:** when using the `:visible` and `:hidden` pseudo-selectors, jQuery tests the actual visibility of the element, not its CSS visibility or display. jQuery looks to see if the element's physical height and width on the page are both greater than zero.

However, this test doesn't work with `<tr>` elements. In the case of `<tr>` jQuery does check the CSS display property, and considers an element hidden if its display property is set to none.

Elements that have not been added to the DOM will always be considered hidden, even if the CSS that would affect them would render them visible. See the [Manipulating Elements](#) section to learn how to create and add elements to the DOM.

## Choosing Selectors

Choosing good selectors is one way to improve JavaScript's performance. A little specificity — for example, including an element type when selecting elements by class name — can go a long way. On the other hand, too much specificity can be a bad thing. A selector such as `#myTable thead tr th.special` is overkill if a selector such as `#myTable th.special` will get the job done.

jQuery offers many attribute-based selectors, allowing selections based on the content of arbitrary attributes using simplified regular expressions.

```
1 // find all <a>s whose rel attribute
2 // ends with "thinger"
3 $("a[rel$='thinger']");
```

While these can be useful in a pinch, they can also be extremely slow in older browsers. Wherever possible, make selections using IDs, class names, and tag names.

## Does My Selection Contain Any Elements?

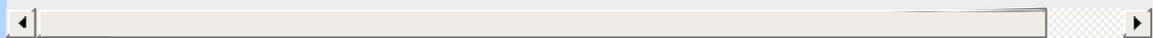
Once you've made a selection, you'll often want to know whether you have anything to work with. A common mistake is to use:

```
1 // Doesn't work!
2 if ( $("div.foo") ) {
3     ...
4 }
```

This won't work. When a selection is made using `$()`, an object is always returned, and objects always evaluate to true. Even if the selection doesn't contain any elements, the code inside the if statement will still run.

The best way to determine if there are any elements is to test the selection's length property, which tells you how many elements were selected. If the answer is 0, the length property will evaluate to false when used as a boolean value:

```
1 //Testing whether a selection contains elements
2 if ( $("#div.foo").length ) {
3     ...
4 }
```



## Saving Selections

jQuery doesn't cache elements for you. If you've made a selection that you might need to make again, you should save the selection in a variable rather than making the selection repeatedly.

```
1 var $divs = $("#div");
```

In the example above, the variable name begins with a dollar sign. Unlike in other languages, there's nothing special about the dollar sign in JavaScript — it's just another character. Here, it's used to indicate that the variable contains a jQuery object. This practice is merely convention, and is not mandatory.

Once the selection is stored in a variable, you can call jQuery methods on the variable just like you would have called them on the original selection.

A selection only fetches the elements that are on the page at the time the selection is made. If elements are added to the page later, you'll have to repeat the selection or otherwise add them to the selection stored in the variable. Stored selections don't magically update when the DOM changes.

## Refining & Filtering Selections

Sometimes the selection contains more than what you're after. jQuery offers several methods for refining and filtering selections.

```
1 // Refining selections
2 $("div.foo").has("p"); // div.foo el
3 $("h1").not(".bar"); // h1 element
4 $("ul li").filter(".current"); // unordered
5 $("ul li").first(); // just the f
6 $("ul li").eq( 5 ); // the sixth
```

## Selecting Form Elements

jQuery offers several pseudo-selectors that help find elements in forms. These are especially helpful because it can be difficult to distinguish between form elements based on their state or type using standard CSS selectors.

### **:button**

Using the `:button` pseudo-selector targets just the `<button>` elements and elements with a `type = "button"`:

```
1 // :button pseudo-selector
2 // selects <button> elements and elements wit
3 $("form :button");
```

In order to get the best performance using `:button`, it's best to first select elements with a standard jQuery selector, then to use `.filter(":button")`. More can be seen on the [jQuery :button documentation page](#). Another option is to precede the pseudo-selector with a tag name or some other selector.

### **:checkbox**

Using the `:checkbox` pseudo-selector targets any elements with a

```
type = "checkbox":
```

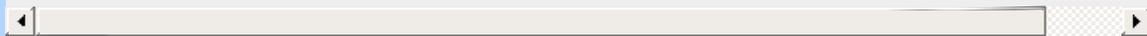
```
1 // :checkbox pseudo-selector
2 // selects elements with type="checkbox"
3 $("form :checkbox");
```

Much like the `:button` pseudo-selector, it's best to first select elements with a standard jQuery selector, then to use `.filter(":checkbox")`, or to precede the pseudo-selector with some other selector.

### **:checked**

Not to be confused with `:checkbox`, `:checked` targets only the *checked* checkboxes as well as checked radio buttons.

```
1 // :checked pseudo-selector
2 // selects checked checkboxes and radio buttons
3 $("form :checked");
```

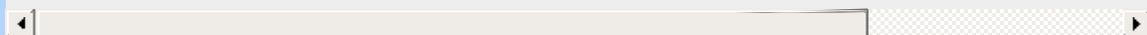


The `:checked` pseudo-selector works when used with **checkboxes** and **radio** buttons.

### **:disabled**

Using the `:disabled` pseudo-selector targets all `<input>` elements with the `disabled` attribute:

```
1 // :disabled pseudo-selector
2 // selects all input elements with the disabled attribute
3 $("form :disabled");
```



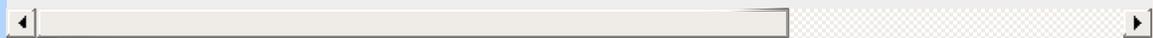
In order to get the best performance using `:disabled`, first select elements with a standard jQuery selector, then use

`.filter(":disabled")`), or precede the pseudo-selector with a tag name or some other selector.

## **:enabled**

Basically the inverse of the *:disabled* pseudo-selector, the `:enabled` pseudo-selector targets all elements that *do not* have a disabled attribute:

```
1 // :enabled pseudo-selector
2 // selects all elements that do not have the
3 $("form :enabled");
```



In order to get the best performance using `:enabled`, first select elements with a standard jQuery selector, then use `.filter(":enabled")`, or precede the pseudo-selector with a tag name or some other selector.

## **:file**

Using the `:file` pseudo-selector targets all `<input>`s that have a `type = "file"`:

```
1 // :file pseudo-selector
2 // selects all inputs with a type = "file"
3 $("form :file");
```

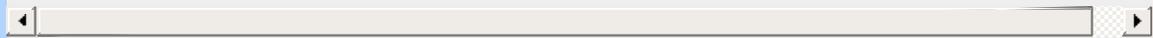
In order to get the best performance using `:file`, first select elements with a standard jQuery selector, then use `.filter(":file")`, or precede the pseudo-selector with a tag name or some other selector.

**Note:** for better performance in modern browsers, use `[type = "file"]` instead of the `:file` pseudo-selector.

## **:image**

Using the `:image` pseudo-selector easily targets all `<input>` tags that are the *image* type:

```
1 // :image pseudo-selector
2 // selects all input elements of type "image"
3 $("form :image");
```



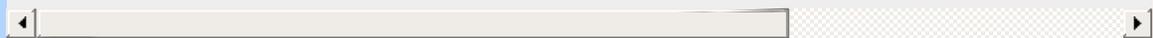
In order to get the best performance using `:image`, first select elements with a standard jQuery selector, then use `.filter(":image")`, or precede the pseudo-selector with a tag name or some other selector.

**Note:** for better performance in modern browsers, use `[type = "image"]` instead of the `:image` pseudo-selector.

## **:input**

Using the `:input` selector selects all `<input>`, `<textarea>`, `<select>`, and `<button>` elements:

```
1 :input pseudo-selector
2 // selects <input>, <textarea>, <select>, and
3 $("form :input");
```



## **:password**

Using the `:password` pseudo-selector easily targets any `<input>`s with a type of *password*:

```
1 // :password pseudo-selector
2 // selects all <input>s "password"
3 $("form :password");
```

In order to get the best performance using `:password`, first select

elements with a standard jQuery selector, then use `.filter(":password")`, or precede the pseudo-selector with a tag name or some other selector.

**Note:** for better performance in modern browsers, use `[ type = "password" ]` instead of the `:password` pseudo-selector.

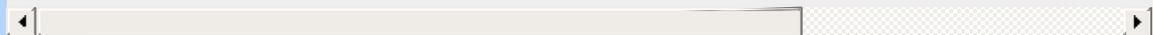
## :radio

Using the `:radio` pseudo-selector easily targets any `<input>`s that have a type of *radio*:

```
1 // :radio pseudo-selector
2 // selects all <input>s of type "radio"
3 $("form :radio");
```

To select a set of associated radio buttons use:

```
1 // Selection associated radio buttons with :r
2 // selects all radio buttons with the name at
3 $("form input[name="gender"]:radio")
```



In order to get the best performance using `:radio`, first select elements with a standard jQuery selector, then use `.filter(":radio")`, or precede the pseudo-selector with a tag name or some other selector.

**Note:** for better performance in modern browsers, use `[ type = "radio" ]` instead of the `:radio` pseudo-selector.

## :reset

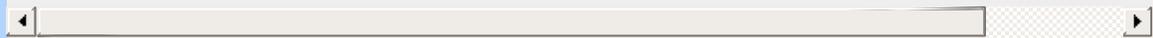
```
1 // :reset pseudo-selector
2 // selects all elements of type "reset"
3 $("form :reset");
```

In order to get the best performance using `:reset`, first select elements with a standard jQuery selector, then use `.filter(":reset")`, or precede the pseudo-selector with a tag name or some other selector.

**Note:** for better performance in modern browsers, use `[ type = "reset" ]` instead of the `:reset` pseudo-selector.

### **:selected**

```
1 // :selected pseudo-selector
2 // selects all selected items in <option> ele
3 $("form :selected");
```



In order to get the best performance using `:selected`, first select elements with a standard jQuery selector, then use `.filter(":selected")`, or precede the pseudo-selector with a tag name or some other selector.

### **:submit**

```
1 // :submit pseudo-selector
2 // selects all inputs with type = "submit"
3 $("form :submit");
```

The `:submit` selector usually applies to `<button>` or `<input>` elements. Some browsers (such as Internet Explorer) do not automatically give the `<button>` element a `type = "submit"` by default.

**Note:** for better performance in modern browsers, use `[ type = "submit" ]` instead of the `:submit` pseudo-selector.

### **:text**

```
1 // :text pseudo-selector
```

```
2 | // selects all inputs with type = "text"
3 | $("form :text");
```

In order to get the best performance using `:selected`, first select elements with a standard jQuery selector, then use `.filter(":selected")`, or precede the pseudo-selector with a tag name or some other selector.

**Note:** As of jQuery 1.5.2, `:text` selects `<input>` elements that have no specified *type* attribute. So, `type = "text"` is implied.

# Working with Selections

## Getters & Setters

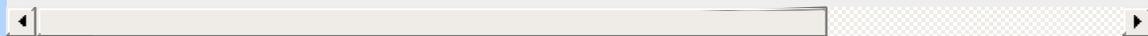
jQuery “overloads” its methods, so the method used to set a value generally has the same name as the method used to get a value. When a method is used to set a value, it's called a setter. When a method is used to get (or read) a value, it's called a getter. Setters affect all elements in a selection. Getters get the requested value only for the first element in the selection.

```
1 //The $.fn.html method used as a setter
2 $("h1").html("hello world");
```

```
1 // The html method used as a getter
2 $("h1").html();
```

Setters return a jQuery object, allowing you to continue calling jQuery methods on your selection. Getters return whatever they were asked to get, so you can't continue to call jQuery methods on the value returned by the getter.

```
1 // Attempting to call a jQuery method after c
2 // This will NOT work
3 $("h1").html().addClass("test")
```



## Chaining

If you call a method on a selection and that method returns a jQuery object, you can continue to call jQuery methods on the object without pausing for a semicolon. This practice is referred to as 'chaining':

```
1 // Chaining
2 $("#content").find("h3").eq( 2 ).html("new te
```

It may help code readability to break the chain over several lines.

```
1 // Formatting chained code
2 $("#content")
3   .find("h3")
4   .eq( 2 )
5   .html("new text for the third h3!");
```

jQuery also provides the `$.fn.end` method to get back to the original selection should you change the selection in the middle of a chain:

```
1 Restoring your original selection using $.fn.
2 $("#content")
3   .find("h3")
4   .eq( 2 )
5   .html("new text for the third h3!")
6   .end() // restores the selection to all h
7   .eq( 0 )
8   .html("new text for the first h3!");
```

Chaining is extraordinarily powerful, and it's a feature that many libraries have adapted since it was made popular by jQuery. However, it must be used with care — extensive chaining can make code extremely difficult to modify or debug. There is no hard-and-fast rule to how long a chain should be — just know that it's easy to get carried away.

# Manipulating Elements

For complete documentation of jQuery manipulation methods, visit the [Manipulation documentation on api.jquery.com](https://api.jquery.com/manipulation/).

## Getting and Setting Information about Elements

There are many ways to change an existing element. Among the most common tasks is changing the inner HTML or attribute of an element. jQuery offers simple, cross-browser methods for these sorts of manipulations. You can also get information about elements using many of the same methods in their getter incarnations. For more information on getters and setters, see the [Working with Selections](#) section. Here are a few methods you can use to get and set information about elements:

- `$.fn.html` - Get or set the html contents.
- `$.fn.text` - Get or set the text contents; HTML will be stripped.
- `$.fn.attr` - Get or set the value of the provided attribute.
- `$.fn.width` - Get or set the width in pixels of the first element in the selection as an integer.
- `$.fn.height` - Get or set the height in pixels of the first element in the selection as an integer.
- `$.fn.position` - Get an object with position information for the first element in the selection, relative to its first positioned ancestor. *This is a getter only.*
- `$.fn.val` - Get or set the value of form elements.

Changing things about elements is trivial, but remember that the change will affect all elements in the selection. If you just want to change one element, be sure to specify that in the selection before calling a setter method.

```
1 // Changing the HTML of an element
2 $("#myDiv p:first").html("New <strong>first</strong>");
```

## Moving, Copying, and Removing Elements

While there are a variety of ways to move elements around the DOM, there are generally two approaches:

- Place the selected element(s) relative to another element.
- Place an element relative to the selected element(s).

For example, jQuery provides `$.fn.insertAfter` and `$.fn.after`. The `$.fn.insertAfter` method places the selected element(s) after the element that provided as an argument. The `$.fn.after` method places the element provided as an argument after the selected element. Several other methods follow this pattern:

`$.fn.insertBefore` and `$.fn.before`, `$.fn.appendTo` and `$.fn.append`, and `$.fn.prependTo` and `$.fn.prepend`.

The method that makes the most sense will depend on what elements are selected, and whether you need to store a reference to the elements you're adding to the page. If you need to store a reference, you will always want to take the first approach — placing the selected elements relative to another element — as it returns the element(s) you're placing. In this case, `$.fn.insertAfter`, `$.fn.insertBefore`, `$.fn.appendTo`, and `$.fn.prependTo` should be the tools of choice.

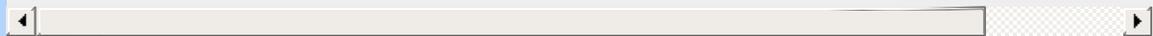
```
1 // Moving elements using different approaches
2
3 // make the first list item the last list item
4 var $li = $("#myList li:first").appendTo("#myList")
5
6 // another approach to the same problem
7 $("#myList").append( $("#myList li:first") )
8
9 // note that there's no way to access the
10 // list item that we moved, as this returns
11 // the list itself
```



## Cloning Elements

Methods such as `$.fn.appendTo` move the element, but sometimes a copy of the element is needed instead. In this case, use `$.fn.clone` first:

```
1 // Making a copy of an element
2
3 // copy the first list item to the end of the
4 $("#myList li:first").clone().appendTo("#myLi
```



If you need to copy related data and events, be sure to pass `true` as an argument to `$.fn.clone`.

## Removing Elements

There are two ways to remove elements from the page:

`$.fn.remove` and `$.fn.detach`. Use `$.fn.remove` when you want to permanently remove the selection from the page. While `$.fn.remove` does return the removed element(s), those elements will not have their associated data and events attached to them if you return them to the page.

Use `$.fn.detach` if you need the data and events to persist. Like `$.fn.remove`, it returns the selection, but it also maintains the data and events associated with the selection, so you can restore the selection to the page at a later time.

The `$.fn.detach` method is extremely valuable if you are doing heavy manipulation on an element. In that case, it's beneficial to `$.fn.detach` the element from the page, work on it in your code, then restore it to the page when you're done. This limits expensive "DOM touches" while maintaining the element's data and events.

If you want to leave the element on the page but remove its contents, you can use `$.fn.empty` to dispose of the element's inner HTML.

## Creating New Elements

jQuery offers a trivial and elegant way to create new elements using the same `$( )` method used to make selections:

```
1 // Creating new elements from an HTML string
2 $("

This is a new paragraph</p>");
3
4 $("-


```

```
1 // Creating a new element with an attribute c
2 $( "<a/>", {
3     html : "This is a <strong>new</strong> li
4     "class" : "new",
5     href : "foo.html"
6 });
```

Note that the attributes object in the second argument above, the property name `class` is quoted, although the property names `'text'` and `'href'` are not. Property names generally do not need to be quoted unless they are reserved words (as `class` is in this case).

When you create a new element, it is not immediately added to the page. There are several ways to add an element to the page once it's been created.

```
1 // Getting a new element on to the page
2 var $myNewElement = $("

New element</p>");
3
4 $myNewElement.appendTo("#content");
5
6 $myNewElement.insertAfter("ul:last"); // this
7


```

```
8 | $("ul").last().after( $myNewElement.clone() );
```

The created element doesn't need to be stored in a variable — you can call the method to add the element to the page directly after the `$( )`. However, most of the time you'll want a reference to the element you added so you won't have to select it later.

You can also create an element as you're adding it to the page, but note that in this case you don't get a reference to the newly created element:

```
1 | // Creating and adding an element to the page
2 | $("ul").append("<li>list item</li>");
```

The syntax for adding new elements to the page is easy, so it's tempting to forget that there's a huge performance cost for adding to the DOM repeatedly. If you're adding many elements to the same container, you'll want to concatenate all the html into a single string, and then append that string to the container instead of appending the elements one at a time. Use an array to gather all the pieces together, then join them into a single string for appending.

```
1 | var myItems = [];
2 | var $myList = $("#myList");
3 |
4 | for ( var i = 0; i < 100; i++ ) {
5 |
6 |     myItems.push( "<li>item " + i + "</li>" );
7 |
8 | }
9 |
10 | $myList.append( myItems.join("") );
```

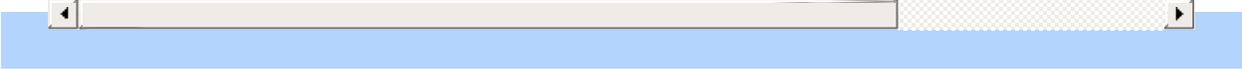
## Manipulating Attributes

jQuery's attribute manipulation capabilities are extensive. Basic changes are simple, but the `$.fn.attr` method also allows for more complex manipulations. It can either set an explicit value, or set a value using the return value of a function. When the function syntax is used, the function receives two arguments: the zero-based index of the element whose attribute is being changed, and the current value of the attribute being changed.

```
1 // Manipulating a single attribute
2 $("#myDiv a:first").attr( "href", "newDestina
```

```
1 // Manipulating multiple attributes
2 $("#myDiv a:first").attr({
3   href: "newDestination.html",
4   rel: "super-special"
5 });
```

```
1 // Using a function to determine an attribute
2 $("#myDiv a:first").attr({
3   rel: "super-special",
4   href: function( idx, href ) {
5     return "/new/" + href;
6   }
7 });
8
9 $("#myDiv a:first").attr( "href", function(
10
11   return "/new/" + href;
12
13 });
```



# The jQuery Object

When creating new elements (or selecting existing ones), jQuery returns the elements in a collection. Many developers new to jQuery assume that this collection is an array. It has a zero-indexed sequence of DOM elements, some familiar array functions, and a `.length` property, after all. Actually, the jQuery object is more complicated than that.

## DOM and DOM elements

The Document Object Model (DOM for short) is a representation of an HTML document. It may contain any number of DOM elements. At a high level, a DOM element can be thought of as a "piece" of a web page. It may contain text and/or other DOM elements. DOM elements are described by a type, such as `<div>`, `<a>`, or `<p>`, and any number of attributes such as `src`, `href`, `class` and so on. For a more thorough description, refer to [the official DOM specification from the W3C](#).

Elements have properties like any JavaScript object. Among these properties are attributes like `tagName` and methods like `appendChild`. These properties are the only way to interact with the web page via JavaScript.

## The jQuery Object

It turns out that working directly with DOM elements can be awkward. The jQuery object defines [many](#) methods to smooth out the experience for developers. Some benefits of the jQuery Object include:

*Compatibility* — The implementation of element methods varies across browser vendors and versions. The following snippet attempts to set the inner HTML of a `<tr>` element stored in `target`:

```
1 | var target = document.getElementById("target")
2 |
3 | target.innerHTML = "<td>Hello <b>World</b>!</td>";
```

This works in many cases, but it will fail in most versions of Internet Explorer. In that case, the [recommended approach](#) is to use pure DOM methods instead. By wrapping the `target` element in a jQuery object, these edge cases are taken care of, and the expected result is achieved in all supported browsers:

```
1 | // Setting the inner HTML with jQuery
2 | var target = document.getElementById("target")
3 |
4 | $( target ).html("<td>Hello <b>World</b>!</td>");
```

*Convenience* — There are also a lot of common DOM manipulation use cases that are awkward to accomplish with pure DOM methods. For instance, inserting an element stored in `newElement` after the `target` element requires a rather verbose DOM method:

```
1 | // Inserting a new element after another with
2 | var target = document.getElementById("target")
```

```
3
4  var newElement = document.createElement("div");
5
6  target.parentNode.insertBefore( target.nextSibling, new
```

By wrapping the `target` element in a jQuery object, the same task becomes much simpler:

```
1  // Inserting a new element after another with
2  var target = document.getElementById("target"
3
4  var newElement = document.createElement("div"
5
6  $( target ).after( newElement );
```

For the most part, these details are simply "gotchas" standing between you and your goals.

## Getting Elements in to the jQuery Object

When the jQuery function is invoked with a CSS selector, it will return a jQuery object wrapping any element(s) that match this selector. For instance, writing:

```
1  // Selecting all 'h1' tags
2  var headers = $("h1");
```

`headers` is now a jQuery element containing *all* the `<h1>` tags already on the page. This can be verified by inspecting the `length` property of `headers`:

```
1  // Viewing the number of 'h1' tags on the pag
2  var allHeaders = $("h1");
```

```
3
4 alert( allHeaders.length );
```

If the page has more than one `<h1>` tag, this number will be greater than one. If the page has no `<h1>` tags, the `length` property will be zero. Checking the `length` property is a common way to ensure that the selector successfully matched one or more elements.

If the goal is to select only the first header element, another step is required. There are a number of ways to accomplish this, but the most straight-forward is the `eq()` function.

```
1 // Selecting only the first 'h1' element on t
2 var headers = $("h1");
3
4 var firstHeader = headers.eq( 0 );
```

Now `firstHeader` is a jQuery object containing only the first `<h1>` element on the page. And because `firstHeader` is a jQuery object, it has useful methods like `.html()` and `.after()`. jQuery also has a method named `.get()` which provides a related function. Instead of returning a jQuery-wrapped DOM element, it returns the DOM element itself.

```
1 // Selecting only the first 'h1' element on t
2 var firstHeaderElem = $("h1").get( 0 );
```

Alternatively, because the jQuery object is "array-like," it supports array subscripting via brackets:

```
1 // Selecting only the first 'h1' element on t
2 var firstHeaderElem = $("h1")[ 0 ];
```

In either case, `firstHeaderElem` contains the native DOM element. This means it has DOM properties like `.innerHTML` and methods like `.appendChild()`, but *not* jQuery methods like `.html()` or `.after()`. The `firstHeaderElem` element is more difficult to work with, but there are certain instances that require it. One such instance is making comparisons.

## Not All jQuery Objects are Created ===

An important detail regarding this "wrapping" behavior is that each wrapped object is unique. This is true *even if the object was created with the same selector or contain references to the exact same DOM elements*.

```
1 // Creating two jQuery objects for the same element
2 var logo1 = $("#logo");
3 var logo2 = $("#logo");
```

Although `logo1` and `logo2` are created in the same way (and wrap the same DOM element), they are not the same object. For example:

```
1 // Comparing jQuery objects
2 alert( $("#logo") === $("#logo") ); // alerts false
```

However, both objects contain the same DOM element. The `get` method is useful for testing if two jQuery objects have the same DOM element.

```
1 // Comparing DOM elements
2 var logo1 = $("#logo");
3 var logo1Elem = logo1.get( 0 );
4
```

```
5 var logo2 = $("#logo");
6 var logo2Elem = logo2.get( 0 );
7
8 alert( logo1Elem === logo2Elem ); // alerts 'true'
```

Many developers prefix a `$` to the name of variables that contain jQuery objects in order to help differentiate. There is nothing magic about this practice — it just helps some people keep track of what different variables contain. The previous example could be re-written to follow this convention:

```
1 // Comparing DOM elements (with more readable
2 var $logo1 = $("#logo");
3 var logo1 = $logo1.get( 0 );
4
5 var $logo2 = $("#logo");
6 var logo2 = $logo2.get( 0 );
7
8 alert( logo1 === logo2 ); // alerts 'true'
```

This code functions identically to the example above, but it is a little more clear to read.

Regardless of the naming convention used, it is very important to make the distinction between jQuery object and native DOM elements. Native DOM methods and properties are not present on the jQuery object, and vice versa. Error messages like "event.target.closest is not a function" and "TypeError: Object [object Object] has no method 'setAttribute'" indicate the presence of this common mistake.

## jQuery Objects Are Not "Live"

Given a jQuery object with all the paragraph elements on the page:

```
1 | // Selecting all 'p' elements on the page
2 | var allParagraphs = $("p");
```

...one might expect that the contents will grow and shrink over time as `<p>` elements are added and removed from the document. jQuery objects do **not** behave in this manner. The set of elements contained within a jQuery object will not change unless explicitly modified. This means that the collection is not "live" — it does not automatically update as the document changes. If the document may have changed since the creation the jQuery object, the collection should be updated by creating a new one. It can be as easy as re-running the same selector:

```
1 | // Updating the selection
2 | allParagraphs = $("p");
```

## Wrapping Up

Although DOM elements provide all the functionality one needs to create interactive web pages, they can be a hassle to work with. The jQuery object wraps these elements to smooth out this experience and make common tasks easy. When creating or selecting elements with jQuery, the result will always be wrapped in a new jQuery object. If the situation calls for the native DOM elements, they may be accessed through the `.get()` method and/or array-style subscripting.

# Traversing

Once you've made an initial selection with jQuery, you can traverse deeper into what was just selected. Traversing can be broken down into three basic parts: parents, children, and siblings. jQuery has an abundance of easy-to-use methods for all these parts. Notice that each of these methods can optionally be passed string selectors, and some can also take another jQuery object in order to filter your selection down. Pay attention and refer to the [API documentation on traversing](#) to know what variation of arguments you have available.

## Parents

The methods for finding the parents from a selection include `$.fn.parent()`, `$.fn.parents()`, `$.fn.parentsUntil()`, and `$.fn.closest()`.

```
1 <div class="grandparent">
2   <div class="parent">
3     <div class="child">
4       <span class="subchild"></span>
5     </div>
6   </div>
7   <div class="surrogateParent1"></div>
8   <div class="surrogateParent2"></div>
9 </div>
```

```
1 // Selecting an element's direct parent
2
3 // returns [ div.child ]
4 $("span.subchild").parent();
5
6 // Selecting all the parents of an element t
7
8 // returns [ div.parent ]
9 $("span.subchild").parents("div.parent");
10
11 // returns [ div.child, div.parent, div.gran
12 $("span.subchild").parents();
13
14 // Selecting all the parents of an element u
15
16 // returns [ div.child, div.parent ]
17 $("span.subchild").parentsUntil("div.grandpa
18
```

```
19 // Selecting the closest parent, note that only one pa
20 // and that the initial element itself is included in
21
22 // returns [ div.child ]
23 $("span.subchild").closest("div");
24 // returns [ div.child ] as the selector is also inclu
25 $("div.child").closest("div");
```



## Children

The methods for finding child elements from a selection include `$.fn.children()` and `$.fn.find()`. The difference between these methods lies in how far into the child structure the selection is made. `$.fn.children()` only operates on direct child nodes, while `$.fn.find()` can traverse recursively into children, children of those children, and so on.

```
1 // Selecting an element's direct children
2
3 // returns [ div.parent, div.surrogateParent1
4 $("div.grandparent").children("div");
5
6 // Finding all elements within a selection th
7
8 // returns [ div.child, div.parent, div.surrc
9 $("div.grandparent").find("div");
```

## Siblings

The rest of the traversal methods within jQuery all deal with finding sibling selections. There are a few basic methods as far as the direction of traversal is concerned. You can find previous elements with `$.fn.prev()`, next elements with `$.fn.next()`, and both with `$.fn.siblings()`. There are also a few other methods that build onto these basic methods: `$.fn.nextAll()`, `$.fn.nextUntil()`, `$.fn.prevAll()` and `$.fn.prevUntil()`.

```
1 // Selecting a next sibling of the selectors
2
3 // returns [ div.surrogateParent1 ]
4 $("div.parent").next();
5
6 // Selecting a prev sibling of the selectors
7
8 // returns [] as No sibling exists before di
9 $("div.parent").prev();
10
11 // Selecting all the next siblings of the se
12
13 // returns [ div.surrogateParent1, div.surro
14 $("div.parent").nextAll();
15 // returns [ div.surrogateParent1 ]
16 $("div.parent").nextAll().first();
17 // returns [ div.surrogateParent2 ]
18 $("div.parent").nextAll().last();
19
20 // Selecting all the previous siblings of th
21
22 // returns [ div.surrogateParent1, div.paren
23 $("div.surrogateParent2").prevAll();
24 // returns [ div.surrogateParent1 ]
25 $("div.surrogateParent2").prevAll().first();
26 // returns [ div.parent ]
```

```
27 | $("div.surrogateParent2").prevAll().last();
```

Use `$.fn.siblings()` to select all siblings:

```
1 | // Selecting an element's siblings in both di
2 |
3 | // returns [ div.surrogateParent1, div.surrog
4 | $("div.parent").siblings();
5 | // returns [ div.parent, div.surrogateParent2
6 | $("div.surrogateParent1").siblings();
```

See the complete documentation for these methods and more at [Traversal documentation on api.jquery.com](#).

Be cautious when traversing long distances in documents — complex traversal makes it imperative that the document's structure remain the same, which is difficult to guarantee even if you're the one creating the whole application from server to client. One- or two-step traversal is fine, but it's best to avoid traversals that go from one container to another.

# CSS, Styling, & Dimensions

jQuery includes a handy way to get and set CSS properties of elements:

```
1 // Getting CSS properties
2 $("h1").css("fontSize"); // returns a string
3 $("h1").css("font-size"); // also works
```

```
1 // Setting CSS properties
2 $("h1").css( "fontSize", "100px" ); // settin
3
4 // setting multiple properties
5 $("h1").css({
6     fontSize: "100px",
7     color: "red"
8 });
```

Note the style of the argument on the second line — it is an object that contains multiple properties. This is a common way to pass multiple arguments to a function, and many jQuery setter methods accept objects to set multiple values at once.

CSS properties that normally include a hyphen need to be camelCased in JavaScript. For example, the CSS property `font-size` is expressed as `fontSize` when used as a property name in JavaScript. However, this does not apply when passing the name of a CSS property to the `$.fn.css()` method as a string — in that case, either the camelCased or hyphenated form will work.

It's not recommended to use `$.fn.css()` as a setter in production-ready code, but when passing in an object to set CSS, CSS properties will be camelCased instead of using a hyphen.

## Using CSS Classes for Styling

As a getter, the `$.fn.css()` method is valuable. However, it should generally be avoided as a setter in production-ready code, because it's generally best to keep presentational information out of JavaScript code. Instead, write CSS rules for classes that describe the various visual states, and then change the class on the element.

```
1 // Working with classes
2 var $h1 = $("h1");
3
4 $h1.addClass("big");
5 $h1.removeClass("big");
6 $h1.toggleClass("big");
7
8 if ( $h1.hasClass("big") ) { ... }
```

Classes can also be useful for storing state information about an element, such as indicating that an element is selected.

## Dimensions

jQuery offers a variety of methods for obtaining and modifying dimension and position information about an element.

The code below shows a brief overview of the dimensions functionality in jQuery. For complete details about jQuery dimension methods, visit the [dimensions documentation on api.jquery.com](http://api.jquery.com/dimensions).

```
1 // Basic dimensions methods
2
3 // sets the width of all H1 elements
4 $("h1").width("50px");
5 // gets the width of the first H1
6 $("h1").width();
7
8
9 // sets the height of all H1 elements
10 $("h1").height("50px");
11
12 // gets the height of the first H1
13 $("h1").height();
14
15
16 // returns an object containing position
17 // information for the first H1 relative to
18 // its "offset (positioned) parent"
19 $("h1").position();
```

# Data Methods

There's often data about an element you want to store with the element. In plain JavaScript, you might do this by adding a property to the DOM element, but you'd have to deal with memory leaks in some browsers. jQuery offers a straightforward way to store data related to an element, and it manages the memory issues for you.

```
1 // Storing and retrieving data related to an
2 $("#myDiv").data( "keyName", { foo: "bar" } )
3
4 // { foo: "bar" }
5 $("#myDiv").data( "keyName" );
```

Any kind of data can be stored on an element. For the purposes of this article, `$.fn.data` will be used to store references to other elements.

For example, you may want to establish a relationship between a list item and a `<div>` that's inside of it. This relationship could be established every single time the list item is touched, but a better solution would be to establish the relationship once, then store a pointer to the `<div>` on the list item using `$.fn.data`:

```
1 // Storing a relationship between elements u
2 $("#myList li").each(function() {
3
4     var $li = $( this );
5     var $div = $li.find("div.content");
6
7     $li.data( "contentDiv", $div );
8
9 });
10
```

```
11 // later, we don't have to find the div again;  
12 // we can just read it from the list item's data  
13 var $firstLi = $("#myList li:first");  
14  
15 $firstLi.data("contentDiv").html("new content")
```

In addition to passing `$.fn.data` a single key-value pair to store data, you can also pass an object containing one or more pairs.

# Utility Methods

jQuery offers several utility methods in the `$` namespace. These methods are helpful for accomplishing routine programming tasks. For a complete reference on jQuery utility methods, visit the [utilities documentation on api.jquery.com](#).

Below are examples of a few of the utility methods:

`$.trim`

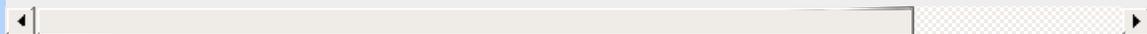
Removes leading and trailing whitespace.

```
1 // returns "lots of extra whitespace"  
2 $.trim("  lots of extra whitespace  ");
```

`$.each`

Iterates over arrays and objects.

```
1 $.each([ "foo", "bar", "baz" ], function( idx  
2     console.log( "element " + idx + "is " + val  
3     });  
4  
5 $.each({ foo: "bar", baz: "bim" }, function(  
6     console.log( k + " : " + v );  
7     });
```



The method `$.fn.each` is also used for iterating over a selection of elements.

`$.inArray`

Returns a value's index in an array, or -1 if the value is not in the array.

```
1 var myArray = [ 1, 2, 3, 5 ];
2
3 if ( $.isArray( 4, myArray ) !== -1 ) {
4     console.log("found it!");
5 }
```

#### `$.extend`

Changes the properties of the first object using the properties of subsequent objects.

```
1 var firstObject = { foo: "bar", a: "b" };
2 var secondObject = { foo: "baz" };
3
4 var newObject = $.extend( firstObject, secondObject );
5
6 console.log( firstObject.foo ); // "bar"
7 console.log( newObject.foo ); // "baz"
```

If you don't want to change any of the objects you pass to `$.extend`, pass an empty object as the first argument.

```
1 var firstObject = { foo: "bar", a: "b" };
2 var secondObject = { foo: "baz" };
3
4 var newObject = $.extend( {}, firstObject, secondObject );
5
6 console.log( firstObject.foo ); // "bar"
7 console.log( newObject.foo ); // "baz"
```

#### `$.proxy`

Returns a function that will always run in the provided scope — that is, sets the meaning of this inside the passed function to the second

argument.

```
1  var myFunction = function() {
2    console.log( this );
3  };
4  var myObject = {
5    foo: "bar"
6  };
7
8  myFunction(); // window
9
10 var myProxyFunction = $.proxy( myFunction, myObject );
11
12 myProxyFunction(); // myObject
```

If you have an object with methods, you can pass the object and the name of a method to return a function that will always run in the scope of the object.

```
1  var myObject = {
2    myFn : function() {
3      console.log( this );
4    }
5  };
6
7  $("#foo").click( myObject.myFn ); // HTMLElement
8  $("#foo").click( $.proxy( myObject, "myFn" )
```

# Iterating over jQuery and non-jQuery Objects

jQuery provides an object iterator utility called `$.each()` as well as a jQuery collection iterator: `.each()`. These are not interchangeable. In addition, there are a couple of helpful methods called `$.map()` and `.map()` that can shortcut one of our common iteration use cases.

`$.each()`

[\\$.each](#) is a generic iterator function for looping over object, arrays, and array-like objects. Plain objects are iterated via their named properties while arrays and array-like objects are iterated via their indices.

`$.each()` is essentially a drop-in replacement of a traditional `for` or 'for-in' loop. Given:

```
1 | var sum = 0;
2 |
3 | var arr = [ 1, 2, 3, 4, 5 ];
```

then this:

```
1 | for ( var i = 0, l = arr.length; i < l; i++ )
2 |     sum += arr[ i ];
3 | }
4 |
5 | console.log( sum ); // 15
```

can be replaced with this:

```
1 | $.each( arr, function( index, value ){
2 |     sum += value;
3 | });
4 |
5 | console.log( sum ); // 15
```

Notice that `arr[ index ]` can't be accessed as the value is conveniently passed to the callback in `$.each()`. In addition, given:

```
1 | var sum = 0;
2 | var obj = {
3 |     foo: 1,
4 |     bar: 2
5 | }
```

then this:

```
1 | for (var item in obj) {
2 |     sum += obj[ item ];
3 | }
4 |
5 | console.log( sum ); // 3
```

can be replaced with this:

```
1 | $.each( obj, function( key, value ) {
2 |     sum += value;
3 | });
4 |
5 | console.log( sum ); // 3
```

Again, `obj[ key ]` is passed directly to the callback and thus can't be accessed. Note that `$.each()` is for plain objects, arrays, array-

like objects *that are not jQuery collections*.

This would be considered incorrect:

```
1 // incorrect
2 $.each( $("p"), function() {
3     // Do something
4 });
```

For jQuery collections, use `.each()`.

`.each()`

`.each()` is used directly on a jQuery collection. It iterates over each matched element in the collection and performs a callback on that object. The index of the current element within the collection is passed as an argument to the callback. The value (the DOM element in this case) is also passed, but the callback is fired within the context of the current matched element so the `this` keyword points to the current element as expected in other jQuery callbacks.

For example, given the following markup:

```
1 <ul>
2   <li><a href="#">Link 1</a></li>
3   <li><a href="#">Link 2</a></li>
4   <li><a href="#">Link 3</a></li>
5 </ul>
```

`.each()` may be used like so:

```
1 $("li").each( function( index, element ){
2     console.log( $( this ).text() );
3 });
4
5 // Logs the following:
```

```
6 // Link 1
7 // Link 2
8 // Link 3
```

## The Second Argument

The question is often raised, "If `this` is the element, why is there a second DOM element argument passed to the callback?"

Whether intentional or inadvertent, the execution context may change. When consistently using the keyword `this`, it's easy to end up confusing ourselves or other developers reading the code. Even if the execution context remains the same, it may be more readable to use the second parameter as a named parameter. For example:

```
1  $("li").each( function( index, listItem ) {
2
3      this === listItem; // true
4
5      // For example only. You probably shouldn't
6      $.ajax({
7          success: function( data ) {
8
9              // The context has changed. The 'this'
10             // no longer refers to listItem.
11             this !== listItem; // true
12         }
13     });
14 });
```

## Sometimes `.each()` Isn't Necessary

Many jQuery methods implicitly iterate over the entire collection, applying their behavior to each matched element. For example, this is unnecessary:

```
1 | $("li").each( function( index, el ) {
2 |     $( el ).addClass("newClass");
3 | });
```

and this is fine:

```
1 | $("li").addClass("newClass");
```

Each `<li/>` in the document will have the class 'newClass' added.

On the other hand, some methods do not iterate over the collection.

`.each()` is required when we need to get information from the element before setting a new value.

This will not work:

```
1 | // doesn't work:
2 | $("input").val( $( this ).val() + "%" );
3 |
4 | // .val() does not change the execution conte
```

Rather, this is how it should be written:

```
1 | $("input").each( function( i, el ) {
2 |     var elem = $( el );
3 |     elem.val( elem.val() + "%" );
4 | });
```

The following is a list of methods that require `.each()`:

- `.attr()` (getter)
- `.css()` (getter)
- `.data()` (getter)

- `.height()` (getter)
- `.html()` (getter)
- `.innerHeight()`
- `.innerWidth()`
- `.offset()` (getter)
- `.outerHeight()`
- `.outerWidth()`
- `.position()`
- `.prop()` (getter)
- `.scrollLeft()` (getter)
- `.scrollTop()` (getter)
- `.val()` (getter)
- `.width()` (getter)

Note that in most cases, the 'getter' signature returns the result from the first element in a jQuery collection while the setter acts over the entire collection of matched elements. The exception to this is `.text()` where the getter signature will return a concatenated string of text from all matched elements.

In addition to a setter value, the attribute, property, css setters, and DOM insertion 'setter' methods (i.e. `.text()` and `.html()`) accept anonymous callback functions that are applied to each element in the matching set. The arguments passed to the callback are the index of the matched element within the set and the result of the 'getter' signature of the method.

For example, these are equivalent:

```
1  $("input").each( function( i, el ) {
2      var elem = $( el );
3      elem.val( elem.val() + "%" );
4  });
5
```

```
6
7   $("input").val(function( index, value ) {
8     return value + "%";
9   });
```

One other thing to keep in mind with this implicit iteration is that traversal methods such as `.children()` or `.parent()` will act on each matched element in a connection, returning a combined collection of all children or parent nodes.

`.map()`

There is a common iteration use case that can be better handled by using the `.map()` method. Anytime we want to create an array or concatenated string based on all matched elements in our jQuery selector, we're better served using `.map()`.

For example instead of doing this:

```
1   var newArr = [];
2
3   $("li").each( function() {
4     newArr.push( this.id );
5   });
```

We can do this:

```
1   $("li").map( function(index, element) {
2     return this.id;
3   }).get();
```

Notice the `.get()` chained at the end. `.map()` actually returns a jQuery-wrapped collection, even if we return strings out of the callback. We need to use the argument-less version of `.get()` in order to return a basic JavaScript array that we can work with. To concatenate into a string, we can chain the plain JS `.join()` array

method after `.get()`.

### `$.map`

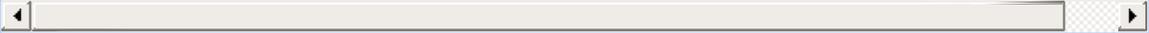
Like `$.each()` and `.each()`, there is a `$.map()` as well as `.map()`. The difference is also very similar to both `.each` methods. `$.map()` works on plain JavaScript arrays while `.map()` works on jQuery element collections. Because it's working on a plain array, `$.map()` returns a plain array and `.get()` does not need to be called — in fact, it will throw an error as it's not a native JavaScript method.

A word of warning: `$.map()` switches the order of callback arguments. This was done in order to match the native JavaScript `.map()` method made available in ECMAScript 5.

For example:

```
1 <li id="a"></li>
2 <li id="b"></li>
3 <li id="c"></li>
4
5 <script>
6
7 var arr = [{
8   id: "a",
9   tagName: "li"
10 }, {
11   id: "b",
12   tagName: "li"
13 }, {
14   id: "c",
15   tagName: "li"
16 }];
17
18
19 // returns [ "a", "b", "c" ]
20 $("li").map( function( index, element ) {
21   return element.id;
22 }).get();
```

```
23
24 // also returns ["a", "b", "c"]
25 // note that the value comes first with $.map
26 $.map( arr, function( value, index ) {
27     return value.id;
28 });
29
30 </script>
```



# Using jQuery's `.index()` Function

`.index()` is a method on jQuery objects that's generally used to search for a given element within the jQuery object that it's called on. This method has four different signatures with different semantics that can be confusing. This article covers details about how to understand the way `.index()` works with each signature.

## `.index()` with No Arguments

```
1 <ul>
2 <div></div>
3 <li id="foo1">foo</li>
4 <li id="bar1">bar</li>
5 <li id="baz1">baz</li>
6 <div></div>
7 </ul>
```

```
1 var $foo = $("#foo1");
2
3 console.log( "Index: " + $foo.index() ); //
4
5 var $listItem = $("li");
6
7 // this implicitly calls .last()
8 console.log( "Index: " + $listItem.index() )
9 console.log( "Index: " + $listItem.last().ir
10
11 var $div = $("div");
12
13 // this implicitly calls .last()
14 console.log( "Index: " + $div.index() ); //
15 console.log( "Index: " + $div.last().index()
```

In the first example, `.index()` gives the zero-based index of `#foo1` within its parent. Since `#foo1` is the second child of its parent, `index()` returns 1.

Potential confusion comes from the other examples of `.index()` in the above code. When `.index()` is called on a jQuery object that contains more than one element, it does not calculate the index of

the first element as might be expected, but instead calculates the index of the last element. This is equivalent to always calling `$jqObject.last().index();`

## `.index()` with a String Argument

```
1 <ul>
2 <div class="test"></div>
3 <li id="foo1">foo</li>
4 <li id="bar1" class="test">bar</li>
5 <li id="baz1">baz</li>
6 <div class="test"></div>
7 </ul>
8 <div id="last"></div>
9
10 var $foo = $("li");
11
12 // this implicitly calls .first()
13 console.log( "Index: " + $foo.index("li") );
14 console.log( "Index: " + $foo.first().index(
15
16 var $baz = $("#baz1");
17 console.log( "Index: " + $baz.index("li"));
18
19 var $listItem = $("#bar1");
20 console.log( "Index: " + $listItem.index(".t
21
22 var $div = $("#last");
23 console.log( "Index: " + $div.index("div") )
```

When `.index()` is called with a string argument, there are two things to consider. First, jQuery will implicitly call `.first()` on the original jQuery object. It will find the index of the first element, not the last element in this case. This is inconsistent, so be careful here.

The second point to consider is that jQuery is querying the entire DOM using the passed in string selector and checking the index within that newly queried jQuery object. For example, when using

`.index("div")` in the last example above, jQuery is selecting all of the `<divs>` in the document, then searching for the index that contains the first element in the jQuery object `.index()` is called on.

## `.index()` with a jQuery Object Argument

```
1 <ul>
2 <div class="test"></div>
3 <li id="foo1">foo</li>
4 <li id="bar1" class="test">bar</li>
5 <li id="baz1">baz</li>
6 <div class="test"></div>
7 </ul>
8 <div id="last"></div>
```

```
1 var $foo = $("li");
2 var $baz = $("#baz1");
3
4 console.log( "Index: " + $foo.index( $baz )
5
6 var $tests = $(".test");
7 var $bar = $("#bar1");
8
9 // implicitly calls .first() on the argument
10 console.log( "Index: " + $tests.index( $bar
11
12 console.log( "Index: " + $tests.index( $bar.
```

In this case, the first element of the jQuery object that is passed into `.index()` is being checked against all of the elements in the original jQuery object. The original jQuery object, on the left side of `.index()`, is array-like and is searched from index 0 through `length - 1` for the first element of the argument jQuery object.

## `.index()` with a DOM Element Argument

In this case, the DOM element that's passed into `.index()` is being checked against all of the elements in the original jQuery object. Once all other cases are understood, this should be the simplest case. It is very similar to the previous case, except since the DOM element is passed directly, it is not taken from a jQuery object container.

# Frequently Asked Questions

- [How do I select an item using class or ID?](#)
- [How do I select elements when I already have a DOM element?](#)
- [How do I test whether an element has a particular class?](#)
- [How do I test whether an element exists?](#)
- [How do I determine the state of a toggled element?](#)
- [How do I select an element by an ID that has characters used in CSS notation?](#)
- [How do I disable/enable a form element?](#)
- [How do I check/uncheck a checkbox input or radio button?](#)
- [How do I get the text value of a selected option?](#)
- [How do I replace text from the 3rd element of a list of 10 items?](#)
- [How do I pull a native DOM element from a jQuery object?](#)

# How do I select an item using class or ID?

This code selects an element with an ID of "myDivId". Since IDs are unique, this expression always selects either zero or one elements depending upon whether or not an element with the specified ID exists.

```
1 | $("#myDivId")
```

This code selects an element with a class of "myCssClass". Since any number of elements can have the same class, this expression will select any number of elements.

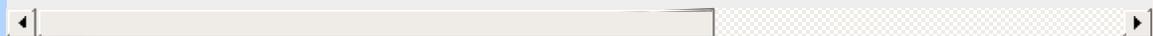
```
1 | $(".myCssClass")
```

A jQuery object containing the selected element can be assigned to a JavaScript variable like normal:

```
1 | var myDivElement = $("#myDivId");
```

Usually, elements in a jQuery object are acted on by other jQuery functions:

```
1 | var myValue = $("#myDivId").val(); // get
2 |
3 | $("#myDivId").val("hello world"); // set
```



## Related Articles

- [Selecting Elements](#)
- [Working with Selections](#)

# How do I select elements when I already have a DOM element?

If you have a variable containing a DOM element, and want to select elements related to that DOM element, simply wrap it in a jQuery object.

```
1 | var myDomElement = document.getElementById("f  
2 |  
3 | $( myDomElement ).find("a"); // finds all anc
```

Many people try to concatenate a DOM element or jQuery object with a CSS selector, like so:

```
1 | $( myDomElement + ".bar" ); // This is equiva
```

Unfortunately, you cannot concatenate strings to objects.

## Related Articles

- [The jQuery Object](#)

# How do I test whether an element has a particular class?

[hasClass](#) (added in version 1.2) handles this common use case:

```
1  $("div").click(function() {
2
3      if ( $( this ).hasClass("protected") ) {
4
5          $( this )
6              .animate({ left: -10 })
7              .animate({ left: 10 })
8              .animate({ left: -10 })
9              .animate({ left: 10 })
10             .animate({ left: 0 });
11
12     }
13
14 });
```

You can also use the [is\(\)](#) method along with an appropriate selector for more advanced matching:

```
1  if ( $("#myDiv").is(".pretty.awesome") ) {
2
3      $("#myDiv").show();
4
5  }
```

Note that this method allows you to test for other things as well. For example, you can test whether an element is hidden (by using the custom [:hidden](#) selector):

```
1  if ( $("#myDiv").is(":hidden") ) {  
2  
3      $("#myDiv").show();  
4  
5  }
```

# How do I test whether an element exists?

Use the [length](#) property of the jQuery collection returned by your selector:

```
1 | if ( $("#myDiv").length ) {  
2 |  
3 |     $("#myDiv").show();  
4 |  
5 | }
```

Note that it isn't always necessary to test whether an element exists. The following code will show the element if it exists, and do nothing (with no errors) if it does not:

```
1 | $("#myDiv").show();
```

# How do I determine the state of a toggled element?

You can determine whether an element is collapsed or not by using the `:visible` and `:hidden` selectors.

```
1 | var isVisible = $("#myDiv").is(":visible");  
2 |  
3 | var isHidden = $("#myDiv").is(":hidden");
```

If you're simply acting on an element based on its visibility, just include `:visible` or `:hidden` in the selector expression. For example:

```
1 | $("#myDiv:visible").animate({  
2 |   left: "+=200px"  
3 | }, "slow" );
```

# How do I select an element by an ID that has characters used in CSS notation?

Because jQuery uses CSS syntax for selecting elements, some characters are interpreted as CSS notation. For example, ID attributes, after an initial letter (a-z or A-Z), may also use periods and colons, in addition to letters, numbers, hyphens, and underscores (see [W3C Basic HTML Data Types](#)). The colon (":") and period (".") are problematic within the context of a jQuery selector because they indicate a pseudo-class and class, respectively.

In order to tell jQuery to treat these characters literally rather than as CSS notation, they must be "escaped" by placing two backslashes in front of them.

```
1 // Does not work
2 $("#some:id")
3
4 // Works!
5 $("#some\\:id")
6
7 // Does not work
8 $("#some.id")
9
10 // Works!
11 $("#some\\.id")
```

The following function takes care of escaping these characters and places a "#" at the beginning of the ID string:

```
1 function jq( myid ) {
```

```
2  
3     return "#" + myid.replace( /(:|\.|\\[|\\])/g, "\\$1"  
4  
5 }
```

The function can be used like so:

```
1 $( jq("some.id") )
```

# How do I disable/enable a form element?

You can enable or disable a form element using the `.prop()` method:

```
1 // Disable #x
2 $("#x").prop( "disabled", true );
3
4 // Enable #x
5 $("#x").prop( "disabled", false );
```

# How do I check/uncheck a checkbox input or radio button?

You can check or uncheck a checkbox element or a radio button using the `.prop()` method:

```
1 // Check #x
2 $("#x").prop( "checked", true );
3
4 // Uncheck #x
5 $("#x").prop( "checked", false );
```

# How do I get the text value of a selected option?

Select elements typically have two values that you want to access. First there's the value to be sent to the server, which is easy:

```
1 | $("#myselect").val();  
2 | // => 1
```

The second is the text value of the select. For example, using the following select box:

```
1 | <select id="myselect">  
2 |   <option value="1">Mr</option>  
3 |   <option value="2">Mrs</option>  
4 |   <option value="3">Ms</option>  
5 |   <option value="4">Dr</option>  
6 |   <option value="5">Prof</option>  
7 | </select>
```

If you wanted to get the string "Mr" if the first option was selected (instead of just "1"), you would do that in the following way:

```
1 | $("#myselect option:selected").text();  
2 | // => "Mr"
```

## How do I replace text from the 3rd element of a list of 10 items?

Either the `:eq()` selector or the `.eq()` method will allow you to select the proper item. However, to replace the text, you must get the value before you set it:

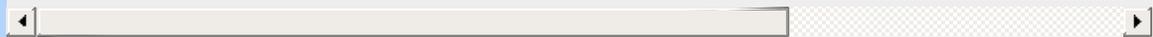
```
1 // This doesn't work; text() returns a string
2 $( this ).find("li a").eq( 2 ).text().replace
3
4 // This works
5 var $thirdLink = $( this ).find("li a").eq( 2
6
7 var linkText = $thirdLink.text().replace( "fc
8
9 $thirdLink.text( linkText );
```

The first example just discards the modified text. The second example saves the modified text and then replaces the old text with the new modified text. Remember, `.text()` gets; `.text("foo")` sets.

# How do I pull a native DOM element from a jQuery object?

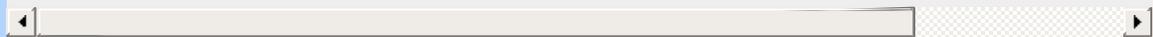
A jQuery object is an array-like wrapper around one or more DOM elements. To get a reference to the actual DOM elements (instead of the jQuery object), you have two options. The first (and fastest) method is to use array notation:

```
1 | $("#foo")[ 0 ]; // equivalent to document.get
```



The second method is to use the [get](#) function:

```
1 | $("#foo").get( 0 ); // identical to above, or
```



You can also call [get](#) without any arguments to retrieve a true array of DOM elements.

# Events

jQuery provides simple methods for attaching event handlers to selections. When an event occurs, the provided function is executed. Inside the function, `this` refers to the element that was clicked.

For details on jQuery events, visit the [Events documentation on api.jquery.com](https://api.jquery.com/events/).

The event handling function can receive an event object. This object can be used to determine the nature of the event, and to prevent the event's default behavior.

For details on the event object, visit the [Event object documentation on api.jquery.com](https://api.jquery.com/event-object/).

- [Event Helpers](#)
- [jQuery Event Basics](#)
- [Introducing Events](#)
- [Handling Events](#)
- [Inside the Event Handling Function](#)
- [Understanding Event Delegation](#)
- [Triggering Event Handlers](#)
- [History of jQuery Events](#)
- [Introducing Custom Events](#)
- [jQuery Event Extensions](#)

# Event Helpers

jQuery offers two event-related helper functions that save you a few keystrokes.

`$.fn.hover`

The `$.fn.hover` method lets you pass one or two functions to be run when the `mouseenter` and `mouseleave` events occur on an element. If you pass one function, it will be run for both events; if you pass two functions, the first will run for `mouseenter`, and the second will run for `mouseleave`.

Prior to jQuery 1.4, the `$.fn.hover` method required two functions.

```
1 // The hover helper function
2 $("#menu li").hover(function() {
3
4     $( this ).toggleClass("hover");
5
6 });
```

`$.fn.toggle`

The `$.fn.toggle` method is triggered by the "click" event and accepts two or more functions. Each time the click event occurs, the next function in the list is called. Generally, `$.fn.toggle` is used with just two functions; however, it will accept an unlimited number of functions. Be careful, though: providing a long list of functions can be difficult to debug.

```
1 // The toggle helper function
2 $("p.expander").toggle( function() {
3     $( this ).prev().addClass("open");
4 }, function() {
5     $( this ).prev().removeClass("open");
6 });
```



# jQuery Event Basics

## jQuery Event Basics

### Setting Up Event Responses on DOM Elements

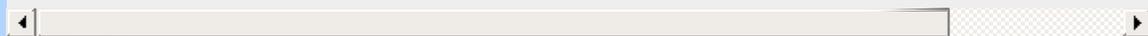
jQuery makes it straightforward to set up event-driven responses on page elements. These events are often triggered by the end user's interaction with the page, such as when text is entered into a form element or the mouse pointer is moved.

In some cases, such as the page load and unload events, the browser itself will trigger the event.

jQuery offers convenience methods for most native browser events. These methods — including `$.fn.click`, `$.fn.focus`, `$.fn.blur`, `$.fn.change`, etc. — are shorthand for jQuery's `$.fn.on` method. The `on` method is useful for binding the same handler function to multiple events, when you want to provide data to the event handler, when you are working with custom events, or when you want to pass an object of multiple events and handlers.

```
1 // Event setup using a convenience method
2 $('p').click(function() {
3     console.log('You clicked a paragraph!');
4 });
```

```
1 // Equivalent event setup using the `$.fn.on`
2 $('p').on('click', function() {
3     console.log('click');
4 });
```



### Extending Events to New Page Elements

It is important to note that `$.fn.on` can only create event listeners on elements that exist *at the time you set up the listeners*. Similar

elements created after the event listeners are established will not automatically pick up event behaviors you've set up previously. For example:

```
1 $(document).ready(function(){
2     // Sets up click behavior on all button e
3     // that exist in the DOM when the instruc
4     $('button.alert').on('click', function(){
5         console.log('A button with the alert c
6     });
7     // Now create a new button element with t
8     // was created after the click listeners
9     // will not have the same click behavior
10    $('button').addClass('alert').appendTo(dc
11    });
```

Consult the article on event delegation to see how to use `$.fn.on` so that event behaviors will be extended to new elements without having to rebind them.

## Inside the Event Handler Function

Every event handling function receives an event object, which contains many properties and methods. The event object is most commonly used to prevent the default action of the event via the `preventDefault` method. However, the event object contains a number of other useful properties and methods, including:

### **pageX, pageY**

The mouse position at the time the event occurred, relative to the top left corner of the page display area (not the entire browser window).

### **type**

The type of the event (e.g. "click").

## which

The button or key that was pressed.

## data

Any data that was passed in when the event was bound. For example:

```
1 // Event setup using the `$.fn.on` method wit
2 $('input').on(
3   'change',
4   {foo : 'bar'}, // associate data with event
5   function(eventObject) {
6     console.log('An input value has changed!
7   }
8 );
```

## target

The DOM element that initiated the event.

## namespace

The namespace specified when the event was triggered.

## timeStamp

The difference in milliseconds between the time the event occurred in the browser and January 1, 1970.

## preventDefault()

Prevent the default action of the event (e.g. following a link).

## stopPropagation()

Stop the event from bubbling up to other elements.

In addition to the event object, the event handling function also has

access to the DOM element that the handler was bound to via the keyword `this`. To turn the DOM element into a jQuery object that we can use jQuery methods on, we simply do `$(this)`, often following this idiom:

```
1 | var $this = $(this);
```

A fuller example would be:

```
1 | // Preventing a link from being followed
2 | $('a').click(function(eventObject) {
3 |     var $this = $(this);
4 |     if ($this.attr('href').match(/evil/)) {
5 |         eventObject.preventDefault();
6 |         $this.addClass('evil');
7 |     }
8 | });
```

## Setting Up Multiple Event Responses

Quite often elements in your application will be bound to multiple events. If multiple events are to share the same handling function, you can provide the event types as a space-separated list to `$.fn.on`:

```
1 | // Multiple events, same handler
2 | $('input').on(
3 |     'click change', // bind listeners for mult
4 |     function() {
5 |         console.log('An input was clicked or char
6 |     }
7 | );
```

When each event has its own handler, you can pass an object into `$.fn.on` with one or more key/value pairs, with the key being the event name and the value being the function to handle the event.

```
1 // Binding multiple events with different handlers
2 $('p').on({
3   'click': function() { console.log('clicked!'); },
4   'mouseover': function() { console.log('hovered!'); },
5 });
```

## Namespacing Events

For complex applications and for plugins you share with others, it can be useful to namespace your events so you don't unintentionally disconnect events that you didn't or couldn't know about.

```
1 // Namespacing events
2 $('p').on('click.myNamespace', function() { console.log('clicked!'); });
3 $('p').off('click.myNamespace');
4 $('p').off('.myNamespace'); // unbind all events in myNamespace
```

## Tearing Down Event Listeners

To remove an event listener, you use the `$.fn.off` method and pass in the event type to off. If you attached a named function to the event, then you can isolate the event tear down to just that named function by passing it as the second argument.

```
1 // Tearing down all click handlers on a selected element
2 $('p').off('click');
```

```
1 // Tearing down a particular click handler, using a ref
2 var foo = function() { console.log('foo'); };
3 var bar = function() { console.log('bar'); };
4
5 $('p').on('click', foo).on('click', bar);
6 $('p').off('click', bar); // foo is still bound to the
```

## Setting Up Events to Run Only Once

Sometimes you need a particular handler to run only once — after that, you may want no handler to run, or you may want a different handler to run. jQuery provides the `$.fn.one` method for this purpose.

```
1 // Switching handlers using the `$.fn.one` me
2 $('p').one('click', firstClick);
3
4 function firstClick(){
5     console.log('You just clicked this for the
6     // Now set up the new handler for subsequ
7     // omit this step if no further click resp
8     $(this).click(function() { console.log('Yc
9 }
```

Note that in the code snippet above, the `firstClick` function will be executed for the first click on *each* paragraph element rather than the function being removed from *all* paragraphs when *any* paragraph is clicked for the first time.

`$.fn.one` can also be used to bind multiple events:

```
1 // Using $.fn.one to bind several events
2 $('input[id]').one('focus mouseover keydown',
```

```
3  
4 function firstEvent(eventObject){  
5     console.log('A ' + eventObject.type + ' event occurred');  
6 }
```

In this case, the `firstEvent` function will be executed once *for each event*. For the snippet above, this means that once an input element gains focus, the handler function will still execute for the first keydown event on that element.

# Introducing Events

## Introduction

Web pages are all about interaction. Users perform a countless number of actions such as moving their mice over the page, clicking on elements, and typing in textboxes—all of these are examples of events. In addition to these user events, there are a slew of others that occur, like when the page is loaded, when video begins playing or is paused, etc. Whenever something interesting occurs on the page, an event is fired, meaning that the browser basically announces that something has happened. It's this announcement that allows developers to "listen" for events and react to them appropriately.

## What's a DOM event?

As mentioned, there are a myriad of event types, but perhaps the ones that are easiest to understand are user events, like when someone clicks on an element or types into a form. These types of events occur on an element, meaning that when a user clicks on a button for example, the button has had an event occur on it. While user interactions aren't the only types of DOM events, they're certainly the easiest to understand when starting out. MDN has a good reference of [available DOM events](#).

## Ways to listen for events

There are many ways to listen for events. Actions are constantly occurring on a webpage, but the developer is only notified about them if they're *listening* for them. Listening for an event basically means you're waiting for the browser to tell you that a specific event has occurred and then you'll specify how the page should react.

To specify to the browser what to do when an event occurs, you provide a function, also known as an *event handler*. This function is executed whenever the event occurs (or until the event is unbound).

For instance, to alert a message whenever a user clicks on a button, you might write something like this:

```
1 | <button onclick="alert('Hello')">Say hello</b>
```

The event we want to listen to is specified by the button's `onclick` attribute, and the event handler is the `alert` function which alerts "Hello" to the user. While this works, it's an abysmal way to achieve this functionality for a couple of reasons:

- 1 First, we're coupling our view code (HTML) with our interaction code (JS). That means that whenever we need to update functionality, we'd have to edit our HTML which is just a bad practice and a maintenance nightmare.
- 2 Second, it's not scalable. If you had to attach this functionality onto numerous buttons, you'd not only bloat the page with a bunch of repetitious code, but you would again destroy maintainability.

Utilizing inline event handlers like this can be considered *obtrusive JavaScript*, but its opposite, *unobtrusive JavaScript* is a much more common way of discussing the topic. The notion of *unobtrusive JavaScript* is that your HTML and JS are kept separate and are therefore more maintainable. Separation of concerns is important because it keeps like pieces of code together (i.e. HTML, JS, CSS)

and unlike pieces of code apart, facilitating changes, enhancements, etc. Furthermore, unobtrusive JavaScript stresses the importance of adding the least amount of cruft to a page as possible. If a user's browser doesn't support JavaScript, then it shouldn't be intertwined into the markup of the page. Also, to prevent naming collisions, JS code should utilize a single namespace for different pieces of functionality or libraries. jQuery is a good example of this, in that the `jQuery` object/constructor (and also the `$` alias to `jQuery`) only utilizes a single global variable, and all of jQuery's functionality is packaged into that one object.

To accomplish the desired task unobtrusively, let's change our HTML a little bit by removing the `onclick` attribute and replacing it with an `id`, which we'll utilize to "hook onto" the button from within a script file.

```
1 <button id="helloBtn">Say hello</button>
```

If we wanted to be informed when a user clicks on that button unobtrusively, we might do something like the following in a separate script file:

```
1 //Event binding using addEventListener
2 var helloBtn = document.getElementById("helloc
3
4 helloBtn.addEventListener( "click", function(
5
6     alert("Hello.");
7
8 }, false );
```

Here we're saving a reference to the button element by calling `getElementById` and assigning its return value to a variable. We then call `addEventListener` and provide an event handler function that will be called whenever that event occurs. While there's nothing wrong with this code as it will work fine in modern browsers, it won't fare

well in versions of IE prior to IE9. This is because Microsoft chose to implement a different method, `attachEvent`, as opposed to the W3C standard `addEventListener`, and didn't get around to changing it until IE9 was released. For this reason, it's beneficial to utilize jQuery because it abstracts away browser inconsistencies, allowing developers to use a single API for these types of tasks, as seen below.

```
1 //Event binding using a convenience method
2 $("#helloBtn").click(function( event ) {
3
4     alert("Hello.");
5
6 });
```

The `$("#helloBtn")` code selects the button element using the `$` (aka `jQuery`) function and returns a jQuery object. The jQuery object has a bunch of methods (functions) available to it, one of them named `click`, which resides in the jQuery object's prototype. We call the `click` method on the jQuery object and pass along an anonymous function event handler that's going to be executed when a user clicks the button, alerting "Hello." to the user.

There are a number of ways that events can be listened for using jQuery:

```
1 //The many ways to bind events with jQuery
2 // Attach an event handler directly to the k
3 // shorthand `click` method.
4 $("#helloBtn").click(function( event ) {
5
6     alert("Hello.");
7
8 });
9
10 // Attach an event handler directly the to k
11 // `bind` method, passing it an event string
```

```
12  $("#helloBtn").bind( "click", function( event ) {
13
14      alert("Hello.");
15
16  });
17
18  // As of jQuery 1.7, attach an event handler directly
19  // using jQuery's `on` method.
20  $("#helloBtn").on( "click", function( event ) {
21
22      alert("Hello.");
23
24  });
25
26  // As of jQuery 1.7, attach an event handler to the `body`
27  // is listening for clicks, and will respond whenever
28  // clicked on the page.
29  $("body").on({
30      click: function( event ) {
31
32          alert("Hello.");
33
34      }
35  }, "button" );
36
37  // An alternative to the previous example, using slight
38  $("body").on( "click", "button", function( event ) {
39
40      alert("Hello.");
41
42  });
```



As of jQuery 1.7, all events are bound via the `on` method, whether you call it directly or whether you use an alias/shortcut method such as `bind` or `click`, which are mapped to the `on` method internally. With this in mind, it's beneficial to use the `on` method because the

others are all just syntactic sugar, and utilizing the `on` method is going to result in faster and more consistent code.

Let's look at the `on` examples from above and discuss their differences. In the first example, a string of `click` is passed as the first argument to the `on` method, and an anonymous function is passed as the second. This looks a lot like the `bind` method before it. Here, we're attaching an event handler directly to `#helloBtn`. If there were any other buttons on the page, they wouldn't alert "Hello" when clicked because the event is only attached to `#helloBtn`.

In the second `on` example, we're passing an object (denoted by the curly braces `{}`), which has a property of `click` whose value is an anonymous function. The second argument to the `on` method is a jQuery selector string of `button`. While examples 1–3 are functionally equivalent, example 4 is different in that the `body` element is listening for click events that occur on *any* `button` element, not just `#helloBtn`. The final example above is exactly the same as the one preceding it, but instead of passing an object, we pass an event string, a selector string, and the callback. Both of these are examples of event delegation, a process by which an element higher in the DOM tree listens for events occurring on its children.

Event delegation works because of the notion of *event bubbling*. For most events, whenever something occurs on a page (like an element is clicked), the event travels from the element it occurred on, up to its parent, then up to the parent's parent, and so on, until it reaches the root element, aka the `window`. So in our table example, whenever a `td` is clicked, its parent `tr` would also be notified of the click, the parent `table` would be notified, the `body` would be notified, and ultimately the `window` would be notified as well. While event bubbling and delegation work well, the delegating element (in our example, the `table`) should always be as close to the delegates as possible so the event doesn't have to travel way up the DOM tree before its handler function is called.

The two main pros of event delegation over binding directly to an element (or set of elements) are performance and the aforementioned event bubbling. Imagine having a large table of 1000 cells and binding to an event for each cell. That's 1000

separate event handlers that the browser has to attach, even if they're all mapped to the same function. Instead of binding to each individual cell though, we could instead use delegation to listen for events that occur on the parent table and react accordingly. One event would be bound instead of 1000, resulting in way better performance and memory management.

The event bubbling that occurs affords us the ability to add cells via AJAX for example, without having to bind events directly to those cells since the parent table is listening for clicks and is therefore notified of clicks on its children. If we weren't using delegation, we'd have to constantly bind events for every cell that's added which is not only a performance issue, but could also become a maintenance nightmare.

## The event object

In all of the previous examples, we've been using anonymous functions and specifying an `event` argument within that function. Let's change it up a little bit.

```
1 //Binding a named function
2 function sayHello( event ) {
3
4     alert("Hello.");
5
6 }
7
8 $("#helloBtn").on( "click", sayHello );
```

In this slightly different example, we're defining a function called `sayHello` and then passing that function into the `on` method instead of an anonymous function. So many online examples show anonymous functions used as event handlers, but it's important to realize that you can also pass defined functions as event handlers as well. This is important if different elements or different events should perform the same functionality. This helps to keep your code DRY.

But what about that `event` argument in the `sayHello` function—what is it and why does it matter? In all DOM event callbacks, jQuery passes an *event object* argument which contains information about the event, such as precisely when and where it occurred, what type of event it was, which element the event occurred on, and a plethora of other information. Of course you don't have to call it `event`; you could call it `e` or whatever you want to, but `event` is a pretty common convention.

If the element has default functionality for a specific event (like a link opens a new page, a button in a form submits the form, etc), that default functionality can be cancelled. This is often useful for AJAX requests. When a user clicks on a button to submit a form via AJAX, we'd want to cancel the button/form's default action (to

submit it to the form's `action` attribute), and we would instead do an AJAX request to accomplish the same task for a more seamless experience. To do this, we would utilize the event object and call its `preventDefault` method. We can also prevent the event from bubbling up the DOM tree using `stopPropagation` so that parent elements aren't notified of its occurrence (in the case that event delegation is being used).

```
1 //Preventing a default action from occurring
2 $("form").on( "submit", function( event ) {
3
4     // Prevent the form's default submission.
5     event.preventDefault();
6
7     // Prevent event from bubbling up DOM tree
8     event.stopPropagation();
9
10    // Make an AJAX request to submit the form
11
12 });
```

When utilizing both `preventDefault` and `stopPropagation` simultaneously, you can instead `return false` to achieve both in a more concise manner, but it's advisable to only `return false` when both are actually necessary and not just for the sake of terseness. A final note on `stopPropagation` is that when using it in delegated events, the soonest that event bubbling can be stopped is when the event reaches the element that is delegating it.

It's also important to note that the event object contains a property called `originalEvent`, which is the event object that the browser itself created. jQuery wraps this native event object with some useful methods and properties, but in some instances, you'll need to access the original event via `event.originalEvent` for instance. This is especially useful for touch events on mobile devices and tablets.

Finally, to inspect the event itself and see all of the data it contains,

you should log the event in the browser's console using `console.log`. This will allow you to see all of an event's properties (including the `originalEvent`) which can be really helpful for debugging.

```
1 //Logging an event's information
2 $("form").on( "submit", function( event ) {
3
4     // Prevent the form's default submission.
5     event.preventDefault();
6
7     // Log the event object for inspectin'
8     console.log( event );
9
10    // Make an AJAX request to submit the form
11
12 });
```

# Handling Events

jQuery provides a method `.on()` to respond to any event on the selected elements. This is called an *event binding*. Although `.on()` isn't the only method provided for event binding, it is a best practice to use this for jQuery 1.7+. To learn more, [read more about the evolution of event binding in jQuery](#).

The on method provides several useful features:

- [Bind any event triggered on the selected elements to an event handler](#)
- [Bind multiple events to one event handler](#)
- [Bind multiple events and multiple handlers to the selected elements](#)
- [Use details about the event in the event handler](#)
- [Pass data to the event handler for custom events](#)
- [Bind events to elements that will be rendered in the future](#)

## Examples

### [Simple event binding](#)

```
1 // When any <p> tag is clicked, we expect to
2 $("p").on( "click", function() {
3     console.log("<p> was clicked");
4 });
```



### [Many events, but only one event handler](#)

Suppose you want to trigger the same event whenever the mouse hovers over or leaves the selected elements. The best practice for this is to use "mouseenter mouseleave". Note the difference

between this and the next example.

```
1 // When a user focuses on or changes any input
2 // bind to multiple events
3 $("div").on( "mouseenter mouseleave", function() {
4     console.log("mouse hovered over or left a div");
5 });
```

### Many events and handlers

Suppose that instead you want different event handlers for when the mouse enters and leaves an element. This is more common than the previous example. For example, if you want to show and hide a tooltip on hover, you would use this.

`.on()` accepts an object containing multiple events and handlers.

```
1 $("div").on({
2     mouseenter: function() {
3         console.log("hovered over a div");
4     },
5     mouseleave: function() {
6         console.log("mouse left a div");
7     },
8     click: function() {
9         console.log("clicked on a div");
10    }
11 });
```

### The event object

Handling events can be tricky. It's often helpful to use the extra information contained in the event object passed to the event handler for more control. To become familiar with the event object, use this code to inspect it in your browser console after you click on

a `<div>` in the page. For a breakdown of the event object, see [Inside the Event Handling Function](#).

```
1 | $("div").on( "click", function( event ) {
2 |     console.log( "event object:" );
3 |     console.dir( event );
4 | });
```

### [Passing data to the event handler](#)

You can pass your own data to the event object.

```
1 | $("p").on( "click", {
2 |     foo: "bar"
3 | }, function( event ) {
4 |     console.log( "event data: " + event.data.foo );
5 | });
```

### [Binding events to elements that don't exist yet](#)

This is called *event delegation*. Here's an example just for completeness, but see the page on [Event Delegation](#) for a full explanation.

```
1 | $("ul").on( "click", "li", function() {
2 |     console.log( "Something in a <ul> was clicked" );
3 | });
```

## Connecting Events to Run Only Once

Sometimes you need a particular handler to run only once — after that, you may want no handler to run, or you may want a different handler to run. jQuery provides the `.one()` method for this purpose.

```
1 // Switching handlers using the `.one()` meth
2 $("p").one( "click", function() {
3     console.log("You just clicked this for the
4     $( this ).click(function() {
5         console.log("You have clicked this before
6     });
7 });
```

The `.one()` method is especially useful if you need to do some complicated setup the first time an element is clicked, but not subsequent times.

`.one()` accepts the same arguments as `on()` which means it supports multiple events to one or multiple handlers, passing custom data and event delegation.

## Disconnecting Events

Although all the fun of jQuery occurs in the `.on()` method, its counterpart is just as important if you want to be a responsible developer. `.off()` cleans up that event binding when you don't need it anymore. Complex user interfaces with lots of event bindings can bog down browser performance, so using the `.off()` method diligently is a best practice to ensure that you only have the event bindings that you need, when you need them.

```
1 // Unbinding all click handlers on a selectio
2 $("p").off("click");
```

```
1 // Unbinding a particular click handler, usi
2 var foo = function() {
3     console.log("foo");
4 };
5
```

```
6  var bar = function() {
7      console.log("bar");
8  };
9
10  $("p").on( "click", foo ).on( "click", bar );
11
12  // foo will stay bound to the click event
13  $("p").off( "click", bar );
```

## Namespacing Events

For complex applications and for plugins you share with others, it can be useful to namespace your events so you don't unintentionally disconnect events that you didn't or couldn't know about. For details, see [Event Namespacing](#).

# Inside the Event Handling Function

Every event handling function receives an event object, which contains many properties and methods. The event object is most commonly used to prevent the default action of the event via the `preventDefault` method. However, the event object contains a number of other useful properties and methods, including:

## **pageX, pageY**

The mouse position at the time the event occurred, relative to the top left of the page.

## **type**

The type of the event (e.g. "click").

## **which**

The button or key that was pressed.

## **data**

Any data that was passed in when the event was bound.

## **target**

The DOM element that initiated the event.

## **preventDefault()**

Prevent the default action of the event (e.g. following a link).

## **stopPropagation()**

Stop the event from bubbling up to other elements.

In addition to the event object, the event handling function also has access to the DOM element that the handler was bound to via the keyword `this`. To turn the DOM element into a jQuery object that we

can use jQuery methods on, we simply do `$( this )`, often following this idiom:

```
1 | var $this = $( this );
```

```
1 | // Preventing a link from being followed
2 | $("a").click(function(e) {
3 |
4 |     var $this = $( this );
5 |
6 |     if ( $this.attr("href").match("evil") ) {
7 |
8 |         e.preventDefault();
9 |
10 |        $this.addClass("evil");
11 |
12 |    }
13 |
14 | });
```

# Understanding Event Delegation

Say you have to add new line items to your page, given the following HTML:

```
1 <html>
2   <body>
3     <div id="container">
4       <ul id="list">
5         <li>
6           <a href="#">Item #1</a>
7         </li>
8         <li>
9           <a href="http://somedomain.com">It
10        </li>
11       <li>
12         <a href="#">Item #3</a>
13       </li>
14       <li>...</li>
15       <li>
16         <a href="http://someotherdomain.cc
17       </li>
18     </ul>
19   </div>
20 </body>
21 </html>
```

We need to attach the same event handler to multiple elements. In this example we want to attach an event that will log the text of the anchor tag to the console whenever it is clicked.

We can attach a direct bind click event to each `<li>` using the `.on()` method, that will alert the text inside of it by doing the

following:

```
1 // attach a directly bound event
2 $("#list a").on( "click", function( event ) {
3     event.preventDefault();
4     console.log( $( this ).text() );
5 });
```

While this works perfectly fine, there are drawbacks. Consider this:

```
1 // add a new element on to our existing list
2 $("#list").append( "<li><a href=\"http://newsi
```

If we were to click our newly added item, nothing would happen. This is because of the directly bound event that we attached previously. Direct events are only attached to elements at the time we called the `.on()` method for our existing collection of `<a>`'s, that is only the `<a>`'s that were found when we call `$( )`

## Event Propagation

Understanding how events propagate is an important factor in being able to leverage Event Delegation. Any time an anchor tag is clicked, a *click* event is fired for the:

- `<a>`
- `<li>`
- `<ul>`
- `<div>`
- `<body>`
- `<html>`
- *document root*

Anytime one of these links is clicked you can think of it as if you were clicking the entire document body. This is called *event bubbling* or *event propagation*.

Since we know how events bubble we can create a delegated event that listens for a specific event to happen on our element

```
1 // attach a delegated event
2 $("#list").on( "click", "a", function( event
3     event.preventDefault();
4     console.log( $( this ).text() );
5 });
```

Notice for the second parameter to the `.on()` method we are telling it which selector to listen for. Now when a *click* event is triggered on our `<ul>`, our delegated event will check to see if the triggering element matches our selector (`"a"`). If it does, our anonymous function will execute. We have now attached a single *click* event listener to our `<ul>` instead of an unknown number of directly bound events on our `<a>`s.

Now lets say that whenever a link is clicked we want to check and see if the `href` attribute starts with "http" and if it does we want to set the `target` attribute to `_blank`.

```
1 // attach a delegated event
2 $("#list").on( "click", "a", function( event
3
4     var $elem = $( this );
5
6     if( $elem.is("[href^=http]") ) {
7
8         $elem.attr( "target", "_blank" );
9
10    }
11
12 });
```

This simply passes the `.is()` method a selector to see if the element's `href` attributes starts with "http". Also we have removed the `event.preventDefault();` statement, this is because we want the default action to happen (which is to following the `href`)

We can actually take this a step further and make our code simpler and more concise by allowing the selector argument to `.on()` do our logic for us.

```
1 // attach a delegated event with a more refin
2 $("#list").on( "click", "a[href^=http]", func
3
4     $( this ).attr( "target", "_blank" );
5
6 });
```

## Summary

Event delegation refers to the process of using event bubbling to handle events at a higher level in the DOM than the element on which the event originated. It allows us to attach a single event listener for elements that exist now or in the future.

# Triggering Event Handlers

jQuery provides a way to trigger the event handlers bound to an element without any user interaction via the `.trigger()` method.

## What handlers can be .trigger()'d

jQuery's event handling system is a layer on top of native browser events. When an event handler is added using `.on("click", function() {...})`, it can be triggered using jQuery's `.trigger("click")` because jQuery stores a reference to that handler when it is originally added. Additionally, it will trigger the javascript inside the "onclick" attribute. The `.trigger()` function cannot be used to mimic native browser events, such as clicking on a file input box or an anchor tag. This is because, there is no event handler attached using jQuery's event system that corresponds to these events.

```
1 | <a href="http://learn.jquery.com">Learn jQuery
```



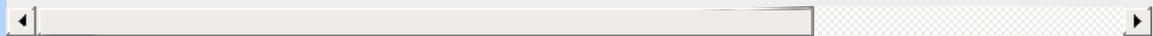
```
1 | //This will not change the current page
2 | $("a").trigger("click");
```

## How can I mimic a native browser event, if not `.trigger()`?

In order to trigger a native browser event, you have to use [document.createEventObject](#) for < IE9 and [document.createEvent](#) for all other browsers. Using these two APIs, you can programmatically create an event that behaves exactly as if someone has actually clicked on a file input box. The default action will happen, and the browse file dialog will display.

The jQuery UI Team created [jquery.simulate.js](#) in order to simplify triggering a native browser event for use in their automated testing. Its usage is modeled after jQuery's `trigger`.

```
1 | //Triggering a native browser event using the
2 | $("a").simulate("click");
```



This will not only trigger the jQuery event handlers, but also follow the link and change the current page.

`.trigger()` **VS** `.triggerHandler()`

There are four differences between `.trigger()` and `.triggerHandler()`

- 1 `.triggerHandler()` only triggers the event on the first element of a jQuery object.
- 2 `.triggerHandler()` cannot be chained. It returns the value that is returned by the last handler, not a jQuery object.
- 3 `.triggerHandler()` will not cause the default behavior of the event (such as a form submission).
- 4 Events triggered by `.triggerHandler()`, will not bubble up the DOM hierarchy. Only the handlers on the single element will fire.

For more information see the [triggerHandler documentation](#)

## Don't use `.trigger()` simply to execute specific functions

While this method has its uses, it should not be used simply to call a function that was bound as a click handler. Instead, you should store the function you want to call in a variable, and pass the variable name when you do your binding. Then, you can call the function itself whenever you want, without the need for `.trigger()`.

```
1 //Triggering an event handler the right way
2 var foo = function( event ) {
3
4     if ( event ) {
5
6         console.log( event );
7
8     } else {
9
10        console.log("this didn't come from an e
11
12    }
13
14 };
15
16 $("p").on( 'click', foo );
17
18 foo(); // instead of $("p").trigger("click")
```

A more complex architecture can be built on top of trigger using the [publish-subscribe pattern](#) using [jQuery plugins](#). With this technique, `$.fn.trigger` can be used to notify other sections of code that an application specific event has happened.

# History of jQuery Events

Throughout the evolution of jQuery the means of event binding has changed for various reasons ranging from performance to semantics. As of jQuery v1.7 the `.on()` method is the accepted means of both directly binding events and creating delegated events. This article aims to explore the history of *event delegation* from jQuery v1.0 - present and how each version leverages it.

Given the following html, for our example we want to log the text of the each `<li>` to console whenever it is clicked.

```
1 <div id="container">
2   <ul id="list">
3     <li>Item #1</li>
4     <li>Item #2</li>
5     <li>Item #3</li>
6     <li>...</li>
7     <li>Item #100</li>
8   </ul>
9 </div>
```

## `.bind()` (Deprecated)

Introduced in jQuery v1.0

It is possible to use `.bind()` and attach a handler to every element.

```
1 $("#list li").bind( "click", function(event)
2   console.log( $elem.text() );
3 });
```

As discussed in the [event delegation](#) article, this is not optimal.

## liveQuery

*liveQuery* was a popular jQuery plugin that allowed for the creation of events which would be triggered for elements that existed now or in the future. This plugin did not use event delegation and used expensive CPU processing to poll the DOM for changes every 20ms and fire events accordingly.

### .bind() delegation (Deprecated)

Introduced in jQuery v1.0

Generally we don't associate `.bind()` with *event delegation*, however prior to jQuery v1.3 it was the only means of delegation available to us.

```
1 | $("#list").bind( "click", function(event) {
2 |     var $elem = $( event.target );
3 |     if ( $elem.is("li") ){
4 |         console.log( $elem.text() );
5 |     }
6 | });
```

We are able to take advantage of *event bubbling* here by attaching a *click* event to the parent `<ul>` element. Whenever the `<li>` is clicked, the event bubbles up to its parent, the `<ul>`, which executes our event handler. Our event handler checks to see if the **event.target** (the element that caused the event to fire) matches our selector.

### .live() (Deprecated)

Introduced in jQuery v1.3

All `.live()` event handlers are bound to the *document* root by default.

```
1 | $("#list li").live( "click", function(event)
```

```
2   var $elem = $( this );
3   console.log( $elem.text() );
4   });
```

When we use `.live()` our event is bound to `$( document )`. When the `<li>` is clicked, bubbling occurs and our `click` event is fired for each of the following elements:

- `<ul>`
- `<div>`
- `<body>`
- `<html>`
- *document root*

The last element to receive the `click` event is *document*, this is where our `.live()` event is bound. `.live()` will then check to see if our selector `#list li` is the element that triggered the event, if so our event handler is executed.

## **.live() w/ context (Deprecated)**

Introduced in jQuery v1.4

Passing the *context* as a second optional argument to the `$( )` function has been supported since v1.0. However support for using this *context* with the `$.live()` method was not added until v1.4.

If we were take our previous `.live()` example and provide it the default *context*, it would look like:

```
1   $( "#list li", document ).live( "click", func
2   var $elem = $( this );
3   console.log( $elem.text() );
4   });
```

Since we can override the *context* when using the `.live()` method, we can specify a *context* that is closer to the element in the DOM hierarchy

```
1 | $( "li", "#list" ).live( "click", function(ev
2 |     var $elem = $( this );
3 |     console.log( $elem.text() );
4 | });
```

In this instance when an `<li>` is clicked the event still bubbles all the way up the *document tree* as it did before. However, our event handler is now bound to the parent `<ul>` tag, so we do not have to wait for the event to bubble all the way up to the *document* root.

## [.delegate\(\)](#) (Deprecated)

First introduced in jQuery v1.4.2

The `.delegate()` method provides a clear difference between the *context* of where to attach delegated event handler, and the *selector* to match when the event bubbles up to the delegated element.

```
1 | $("#list").delegate( "li", "click", function(
2 |     var $elem = $( this );
3 |     console.log( $elem.text() );
4 | });
```

## [.on\(\)](#)

First introduced in jQuery v1.7

The `on.()` method gives us a semantic approach for creating directly bound events as well as delegated events. It eliminates the need to use the deprecated `.bind()`, `.live()` and `.delegate()`

methods, providing a single API for creating events.

```
1 | $("#list").on( "click", "li", function(event)
2 |     var $elem = $( this );
3 |     console.log( $elem.text() );
4 | });
```

## Summary

All of these ways of *event delegation* were innovative and considered a best practice at the time of their release. Depending on what version of jQuery you have implemented use the appropriate means of *event delegation*.

# Introducing Custom Events

## Custom Events

We're all familiar with the basic events — click, mouseover, focus, blur, submit, etc. — that we can latch on to as a user interacts with the browser. Custom events open up a whole new world of event-driven programming. In this chapter, we'll use jQuery's custom events system to make a simple Twitter search application.

It can be difficult at first to understand why you'd want to use custom events, when the built-in events seem to suit your needs just fine. It turns out that custom events offer a whole new way of thinking about event-driven JavaScript. Instead of focusing on the element that triggers an action, custom events put the spotlight on the element being acted upon. This brings a bevy of benefits, including:

- Behaviors of the target element can easily be triggered by different elements using the same code.
- Behaviors can be triggered across multiple, similar, target elements at once.
- Behaviors are more clearly associated with the target element in code, making code easier to read and maintain.

Why should you care? An example is probably the best way to explain. Suppose you have a lightbulb in a room in a house. The lightbulb is currently turned on, and it's controlled by two three-way switches and a clapper:

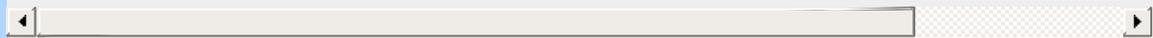
```
1 <div class="room" id="kitchen">
2   <div class="lightbulb on"></div>
3   <div class="switch"></div>
4   <div class="switch"></div>
5   <div class="clapper"></div>
6 </div>
```

Triggering the clapper or either of the switches will change the state of the lightbulb. The switches and the clapper don't care what state

the lightbulb is in; they just want to change the state.

Without custom events, you might write some code like this:

```
1  $(".switch, .clapper").click(function() {
2
3      var $light = $( this ).parent().find(".lig
4
5      if ( $light.hasClass("on") ) {
6
7          $light.removeClass("on").addClass("off"
8
9      } else {
10
11         $light.removeClass("off").addClass("on"
12
13     }
14
15 });
```



With custom events, your code might look more like this:

```
1  $(".lightbulb").on( "changeState", function(
2
3      var $light = $( this );
4
5      if ( $light.hasClass("on") ) {
6
7          $light.removeClass("on").addClass("off
8
9      } else {
10
11         $light.removeClass("off").addClass("or
12
13     }
```

```
14
15     });
16
17     $(".switch, .clapper").click(function() {
18
19         $( this ).parent().find(".lightbulb").trigger(
20
21     });
```

This last bit of code is not that exciting, but something important has happened: we've moved the behavior of the lightbulb to the lightbulb, and away from the switches and the clapper.

Let's make our example a little more interesting. We'll add another room to our house, along with a master switch, as shown here:

```
1 <div class="room" id="kitchen">
2   <div class="lightbulb on"></div>
3   <div class="switch"></div>
4   <div class="switch"></div>
5   <div class="clapper"></div>
6 </div>
7 <div class="room" id="bedroom">
8   <div class="lightbulb on"></div>
9   <div class="switch"></div>
10  <div class="switch"></div>
11  <div class="clapper"></div>
12 </div>
13 <div id="master_switch"></div>
```

If there are any lights on in the house, we want the master switch to turn all the lights off; otherwise, we want it to turn all lights on. To accomplish this, we'll add two more custom events to the lightbulbs: `turnOn` and `turnOff`. We'll make use of them in the `changeState` custom event, and use some logic to decide which one the master switch should trigger:

```
1  $(".lightbulb").on( "changeState", function(
2
3      var $light = $( this );
4
5      if ($light.hasClass("on")) {
6
7          $light.trigger("turnOff");
8
9      } else {
10
11          $light.trigger("turnOn");
12      }
13
14  }).on( "turnOn", function(e) {
15
16      $( this ).removeClass("off").addClass("on"
17
18  }).on( "turnOff", function(e) {
19
20      $( this ).removeClass("on").addClass("off"
21
22  });
23
24  $(".switch, .clapper").click(function() {
25
26      $( this ).parent().find(".lightbulb").trig
27
28  });
29
30  $("#master_switch").click(function() {
31
32      if ( $(".lightbulb.on").length ) {
33
34          $(".lightbulb").trigger("turnOff");
35
36      } else {
```

```
37
38     $(".lightbulb").trigger("turnOn");
39
40     }
41
42     });
```

Note how the behavior of the master switch is attached to the master switch; the behavior of a lightbulb belongs to the lightbulbs.

If you're accustomed to object-oriented programming, you may find it useful to think of custom events as methods of objects. Loosely speaking, the object to which the method belongs is created via the jQuery selector. Binding the `changeState` custom event to all `$(".light")` elements is akin to having a class called `Light` with a method of `changeState`, and then instantiating new `Light` objects for each element with a classname of `light`.

## Recap: `$.fn.on` and `$.fn.trigger`

In the world of custom events, there are two important jQuery methods: `$.fn.on` and `$.fn.trigger`. In the Events chapter, we saw how to use these methods for working with user events; for this chapter, it's important to remember two things:

- `$.fn.on` method takes an event type and an event handling function as arguments. Optionally, it can also receive event-related data as its second argument, pushing the event handling function to the third argument. Any data that is passed will be available to the event handling function in the `data` property of the event object. The event handling function always receives the event object as its first argument.
- `$.fn.trigger` method takes an event type as its argument. Optionally, it can also take an array of values. These values will be passed to the event handling function as arguments after the event object.

Here is an example of the usage of `$.fn.on` and `$.fn.trigger` that

uses custom data in both cases:

```
1 $( document ).on( "myCustomEvent", {
2   foo: "bar"
3 }, function( event, arg1, arg2 ) {
4   console.log( event.data.foo ); // "bar"
5   console.log( arg1 );           // "bim"
6   console.log( arg2 );           // "baz"
7 });
8
9 $( document ).trigger( "myCustomEvent", [ "bi
```

## A Sample Application

To demonstrate the power of custom events, we're going to create a simple tool for searching Twitter. The tool will offer several ways for a user to add search terms to the display: by entering a search term in a text box, by entering multiple search terms in the URL, and by querying Twitter for trending terms.

The results for each term will be shown in a results container; these containers will be able to be expanded, collapsed, refreshed, and removed, either individually or all at once.

When we're done, it will look like this:

# Twitter Search

Load Trending Terms

Refresh All Results

Remove All Results

Collapse All Results

Expand All Results

#gha

Add Search Term

Refresh Remove Collapse

## Search Results for #gha

**Nenevele:** @milky868(God I didnt go to church cos of the #Gha match)..na Advance hell I go enter, make I enter room...peeping though..enjoy the Victory

Sat, 26 Jun 2010 21:32:01 +0000

**jagsinghb:** @Blanconeri10 what did you think of #usa vs #gha.-- who do reckon will winning #worldcup .. as it stands??

Sat, 26 Jun 2010 21:32:00 +0000

**KaVilIela:** RT @marcelotas: Torci muito pros #USA continuar na Copa. Mas é muito legal ver representante africano com a categoria e força de #GHA

Sat, 26 Jun 2010 21:32:00 +0000

**Newsrub:** SpiesList: HuffingtonPost: #USA #GHA in extra time -- LIVE blog, tweets, photos + more  
<http://huff.to/c8GXzM>: Huff... <http://bit.ly/cThHCD>

Sat, 26 Jun 2010 21:32:00 +0000

**DamarisPlat:** Mals uma 'zebra' na copa! Isso ai, camisa não ganha jogo! #gha

Sat, 26 Jun 2010 21:31:59 +0000

Refresh Remove Collapse

## Search Results for #usa

**iamMATT:** RT @jalenrose: RT @wingoz: (#USA loses to Ghana 2-1)Now I'm depressed....ME too!

Sat, 26 Jun 2010 21:31:53 +0000

**patrickryan:** @nod #USA 1st half was sloppy, #GHA took advantage, any team will waste time if they are up in 2nd half, part of the game.

Sat, 26 Jun 2010 21:31:53 +0000

**MatthewDiffee:** I can appreciate the reasons for being happy for Ghana, but maybe wait a few minutes before jumping ship #worldcup #usa

Sat, 26 Jun 2010 21:31:53 +0000

**dlgressus:** I feel like I just broke up with my girlfriend or something. #abouttocry #USA #worldcup

Sat, 26 Jun 2010 21:31:52 +0000

**mlssalazar:** team #usa...so proud of those guys. grateful to them for putting their hearts and souls into the game. #usa #usa #usa

Sat, 26 Jun 2010 21:31:50 +0000

```
1 <h1>Twitter Search</h1>
2 <input type="button" id="get_trends" value="
3
4 <form>
5   <input type="text" class="input_text" id="
6   <input type="submit" class="input_submit"
7 </form>
```

```
8
9 <div id="twitter">
10   <div class="template results">
11     <h2>Search Results for
12     <span class="search_term"></span></h2>
13   </div>
14 </div>
```

This gives us a container (#twitter) for our widget, a template for our results containers (hidden via CSS), and a simple form where users can input a search term. (For the sake of simplicity, we're going to assume that our application is JavaScript-only and that our users will always have CSS.)

There are two types of objects we'll want to act on: the results containers, and the Twitter container.

The results containers are the heart of the application. We'll create a plugin that will prepare each results container once it's added to the Twitter container. Among other things, it will bind the custom events for each container and add the action buttons at the top right of each container. Each results container will have the following custom events:

- `refresh` - Mark the container as being in the "refreshing" state, and fire the request to fetch the data for the search term.
- `populate` - Receive the returned JSON data and use it to populate the container.
- `remove` - Remove the container from the page after the user verifies the request to do so. Verification can be bypassed by passing true as the second argument to the event handler. The remove event also removes the term associated with the results container from the global object containing the search terms.
- `collapse` - Add a class of collapsed to the container, which will hide the results via CSS. It will also turn the container's "Collapse" button into an "Expand" button.

- `expand` - Remove the collapsed class from the container. It will also turn the container's "Expand" button into a "Collapse" button.

The plugin is also responsible for adding the action buttons to the container. It binds a click event to each action's list item, and uses the list item's class to determine which custom event will be triggered on the corresponding results container.

```
1  $.fn.twitterResult = function( settings ) {
2    return this.each(function() {
3      var $results = $( this );
4      var $actions = $.fn.twitterResult.actions;
5      var $a = $actions.clone().prependTo( $results );
6      var term = settings.term;
7
8      $results.find("span.search_term").text(
9        $.each([ "refresh", "populate", "remove" ],
10         function( i, action ) {
11           $results.on( ev, {
12             term: term
13           }, $.fn.twitterResult.events[ ev ] );
14         }
15       ));
16
17       // use the class of each action to figure out
18       // which event it will trigger on the results
19       $a.find("li").click(function() {
20
21         // pass the li that was clicked to the results
22         // so it can be manipulated if needed
23         $results.trigger( $( this ).attr("class") );
24       });
25
26     });
27 }
```

```

27
28 };
29
30 $.fn.twitterResult.createActions = function() {
31     return $("<ul class='actions' />").append(
32         "<li class='refresh'>Refresh</li>" +
33         "<li class='remove'>Remove</li>" +
34         "<li class='collapse'>Collapse</li>"
35     );
36 };
37
38 $.fn.twitterResult.events = {
39
40     refresh: function( e ) {
41         // indicate that the results are refreshing
42         var $this = $( this ).addClass("refreshing"
43
44         $this.find("p.tweet").remove();
45         $results.append("<p class='loading'>Loading ...</
46
47         // get the twitter data using jsonp
48         $.getJSON("http://search.twitter.com/search.json?
49             $this.trigger( "populate", [ json ] );
50     });
51 },
52 populate: function( e, json ) {
53     var results = json.results;
54     var $this = $( this );
55
56     $this.find("p.loading").remove();
57     $.each( results, function( i, result ) {
58         var tweet = "<p class='tweet'>" +
59             "<a href='http://twitter.com/' +
60             result.from_user +
61             "'>" +
62             result.from_user +
63             "</a>: " +

```

```
64         result.text +
65         " <span class='date'>" +
66         result.created_at +
67         "</span>" +
68         "</p>";
69
70     $this.append( tweet );
71 });
72
73     // indicate that the results
74     // are done refreshing
75     $this.removeClass("refreshing");
76 },
77 remove: function( e, force ) {
78     if ( !force && !confirm("Remove panel for term "
79         return;
80     }
81     $( this ).remove();
82
83     // indicate that we no longer have a panel for th
84     search_terms[ e.data.term ] = 0;
85 },
86 collapse: function( e ) {
87     $( this ).find("li.collapse")
88         .removeClass("collapse")
89         .addClass("expand")
90         .text("Expand");
91
92     $( this ).addClass("collapsed");
93 },
94
95 expand: function( e ) {
96     $( this ).find("li.expand")
97         .removeClass("expand")
98         .addClass("collapse")
99         .text("Collapse");
100
```

```
101     $( this ).removeClass("collapsed");
102     }
103
104     };
```

The Twitter container itself will have just two custom events:

- `getResults` - Receives a search term and checks to determine whether there's already a results container for the term; if not, adds a results container using the results template, set up the results container using the `$.fn.twitterResult` plugin discussed above, and then triggers the `refresh` event on the results container in order to actually load the results. Finally, it will store the search term so the application knows not to re-fetch the term.
- `getTrends` - Queries Twitter for the top 10 trending terms, then iterates over them and triggers the `getResults` event for each of them, thereby adding a results container for each term.

Here's how the Twitter container bindings look:

```
1  $("#twitter").on( "getResults", function( e,
2
3      // make sure we don't have a box for this
4      if ( !search_terms[ term ] ) {
5          var $this = $( this );
6          var $template = $this.find("div.template
7
8          // make a copy of the template div
9          // and insert it as the first results bc
10         $results = $template.clone()
11             .removeClass("template")
12             .insertBefore( $this.find("div:first")
13             .twitterResult({
14                 "term": term
```

```

15     });
16
17     // load the content using the "refresh"
18     // custom event that we bound to the results conta
19     $results.trigger("refresh");
20
21     search_terms[ term ] = 1;
22 }
23 }).on( "getTrends", function( e ) {
24     var $this = $( this );
25
26     $.getJSON( "http://search.twitter.com/trends.json?ca
27     var trends = json.trends;
28
29     $.each( trends, function( i, trend ) {
30         $this.trigger( "getResults", [ trend.name ] );
31     });
32 });
33 });

```

So far, we've written a lot of code that does approximately nothing, but that's OK. By specifying all the behaviors that we want our core objects to have, we've created a solid framework for rapidly building out the interface.

Let's start by hooking up our text input and the "Load Trending Terms" button. For the text input, we'll capture the term that was entered in the input and pass it as we trigger the Twitter container's `getResults` event. Clicking the "Load Trending Terms" will trigger the Twitter container's `getTrends` event:

```

1  $("form").submit(function( event ) {
2      var term = $("#search_term").val();
3      $("#twitter").trigger( "getResults", [ ter
4
5      event.preventDefault();

```

```
6   });  
7  
8   $("#get_trends").click(function() {  
9     $("#twitter").trigger("getTrends");  
10  });
```

By adding a few buttons with the appropriate IDs, we can make it possible to remove, collapse, expand, and refresh all results containers at once, as shown below. For the remove button, note how we're passing a value of true to the event handler as its second argument, telling the event handler that we don't want to verify the removal of individual containers.

```
1   $.each([ "refresh", "expand", "collapse" ],  
2     $( "#" + ev ).click( function( e ) {  
3       $("#twitter div.results").trigger( ev );  
4     }));  
5   });  
6  
7   $("#remove").click(function( e ) {  
8     if ( confirm("Remove all results?" ) ) {  
9       $("#twitter div.results").trigger( "remc  
10  }  
11  });
```

## Conclusion

Custom events offer a new way of thinking about your code: they put the emphasis on the target of a behavior, not on the element that triggers it. If you take the time at the outset to spell out the pieces of your application, as well as the behaviors those pieces need to exhibit, custom events can provide a powerful way for you to "talk" to those pieces, either one at a time or en masse. Once the behaviors of a piece have been described, it becomes trivial to trigger those behaviors from anywhere, allowing for rapid creation

of and experimentation with interface options. Finally, custom events can enhance code readability and maintainability, by making clear the relationship between an element and its behaviors.

# jQuery Event Extensions

jQuery offers several ways to extend its event system to provide custom functionality when events are attached to elements. Internally in jQuery, these extensions are primarily used to ensure that standard events such as `submit` and `change` behave consistently across browsers. However, they can also be used to define new events with custom behavior.

This document covers the extensions available starting with jQuery 1.7; a sparsely documented subset of this functionality has been available since jQuery 1.3 but the differences in functionality are extensive. For an overview of special events in earlier versions, see [Ben Alman's jQuery Special Events](#) article.

**Note:** *jQuery event extensions are an advanced feature; they require deeper knowledge of both browser behavior and jQuery internals than most of the API. Most users of jQuery will not need to use event extensions, and those who do should use them with care. For example, on a large project with third-party plugins, changing the behavior of standard events such as `click` or `mouseover` can cause serious compatibility issues.*

## Events overview and general advice

When writing an event extension, it is essential to understand the flow of events through jQuery's internal event system. For a description of the event system from the API level, including a

discussion of event delegation, see the `.on()` method.

To simplify event management, jQuery only attaches a single event handler per element per event type (using `addEventListener` on W3C-compliant browsers or `attachEvent` on older IE) and then dispatches to event handlers that are attached via jQuery's APIs. For example, if three "click" event handlers are attached to an element, jQuery attaches its own handler when the first handler is attached and adds the user's event handler to a list to be executed when the event occurs. For subsequent event handlers, jQuery only adds them to its own internal list since it has already called the browser to attach its solitary handler. Conversely, jQuery removes its own event handler from the browser when the final event of a particular type is removed from the element.

An event can be a *native* event defined by the W3C and fired by the browser in response to something such as a user clicking a mouse button or pressing a key. It can also be a *custom* event, triggered only by code via jQuery's `.trigger()` or `.triggerHandler()` methods. Code can also trigger native browser events, which is convenient for simulating user actions.

In general, jQuery does not have intrinsic knowledge of whether an event name may be fired by a browser. So by default, jQuery always attaches an event to the browser when an API call is made to add an event handler for that event. If that event type is never generated by the browser, the only way the handler will be called is if JavaScript code triggers the event. Although there is generally no harm in attaching an unused event name to the browser, the default behavior can be overridden using the special event `setup` hook as described below.

Whenever elements are removed from a document via jQuery, the event system tries to ensure that events and related data are removed from the elements to prevent memory leaks. (Older versions of Internet Explorer are notorious for leaking memory in these situations if not managed carefully.) If an event extension attaches events or creates new objects, it should detach those objects or clear the data when the event is removed by defining `remove` and `teardown` hooks.

jQuery event extension developers should avoid using event names

that have special meaning in a DOM setting. Event names such as "click", "change", or "load" have specific semantics defined by the W3C, so using them as custom events can cause unexpected behavior. Generally, jQuery event extensions should *only* be used for W3C-defined event names when the extension is normalizing behavior across browsers. A common convention to avoid collisions for custom events is to embed a colon or dash in the event type name, since no W3C events use those characters.

Although jQuery's event system is oriented towards delivering DOM events to DOM elements, jQuery methods can be used to attach and trigger events on plain objects. For example, it can be used as a simple publish/subscribe mechanism. Developers of event extensions should attempt to avoid unwanted behavior if their extensions are used in a mixed scenario with DOM and plain objects. The canonical way that jQuery detects a DOM element is to check for `elem.nodeType === 1` on the object.

## jQuery.event.props: Array

jQuery defines an [Event object](#) that represents a cross-browser subset of the information available when an event occurs. The `jQuery.event.props` property is an array of string names for properties that are always copied when jQuery processes a *native* browser event. (Events fired in code by `.trigger()` do not use this list, since the code can construct a `jQuery.Event` object with the needed values and trigger using that object.)

To add a property name to this list, use

`jQuery.event.props.push("newPropertyName")`. However, be aware that every event processed by jQuery will now attempt to copy this property name from the native browser event to jQuery's constructed event. If the property does not exist for that event type, it will get an undefined value. Adding many properties to this list can significantly reduce event delivery performance, so for infrequently-needed properties it is more efficient to use the value directly from `event.originalEvent` instead. If properties must be copied, you are strongly advised to use `jQuery.event.fixHooks` as of version 1.7.

## jQuery.event.fixHooks: Object

The `fixHooks` interface provides a per-event-type way to extend or normalize the event object that jQuery creates when it processes a *native* browser event. A `fixHooks` entry is an object that has two properties, each being optional:

`props`: Array : Strings representing properties that should be copied from the browser's event object to the jQuery event object. If omitted, no additional properties are copied beyond the standard ones that jQuery copies and normalizes (e.g., `event.target` and `event.relatedTarget`).

`filter`: Function( `event`, `originalEvent` ) : jQuery calls this function after it constructs the `jquery.Event` object, copies standard properties from `jquery.event.props`, and copies the `fixHooks`-specific props (if any) specified above. The function can create new properties on the event object or modify existing ones. The second argument is the browser's native event object, which is also available in `event.originalEvent`.

Note that for all events, the browser's native event object is available in `event.originalEvent`; if the jQuery event handler examines the properties there instead of jQuery's normalized `event` object, there is no need to create a `fixHooks` entry to copy or modify the properties.

For example, to set a hook for the "drop" event that copies the "dataTransfer" property, assign an object to

`jquery.event.fixHooks.drop`:

```
1 | jquery.event.fixHooks.drop = {  
2 |   props: [ "dataTransfer" ]  
3 | };
```

Since `fixHooks` are an advanced feature and rarely used externally, jQuery does not include code or interfaces to deal with conflict resolution. If there is a chance that some other code may be assigning `fixHooks` to the same events, the code should check for an existing hook and take appropriate measures. A simple solution might look like this:

```
1  if ( jQuery.event.fixHooks.drop ) {
2      throw new Error("Someone else took the jQuery.event.f
3  }
4
5  jQuery.event.fixHooks.drop = {
6      props: [ "dataTransfer" ]
7  };
```

When there are known cases of different plugins wanting to attach to the drop hook, this solution might be more appropriate:

```
1  var existingHook = jQuery.event.fixHooks.drc
2
3  if ( !existingHook ) {
4      jQuery.event.fixHooks.drop = {
5          props: [ "dataTransfer" ]
6      };
7  } else {
8      if ( existingHook.props ) {
9          existingHook.props.push( "dataTransfer"
10     } else {
11         existingHook.props = [ "dataTransfer" ];
12     }
13 }
```

## Special event hooks

The jQuery special event hooks are a set of per-event-name functions and properties that allow code to control the behavior of event processing within jQuery. The mechanism is similar to `fixHooks` in that the special event information is stored in `jQuery.event.special.NAME`, where `NAME` is the name of the special event. Event names are case sensitive.

As with `fixHooks`, the special event hooks design assumes it will be very rare that two unrelated pieces of code want to process the same event name. Special event authors who need to modify events with existing hooks will need to take precautions to avoid introducing unwanted side-effects by clobbering those hooks.

### **noBubble: Boolean**

Indicates whether this event type should be bubbled when the `.trigger()` method is called; by default it is `false`, meaning that a triggered event will bubble to the element's parents up to the document (if attached to a document) and then to the window. Note that defining `noBubble` on an event will effectively prevent that event from being used for delegated events with `.trigger()`.

### **bindType: String, delegateType: String**

When defined, these string properties specify that a special event should be handled like another event type until the event is delivered. The `bindType` is used if the event is attached directly, and the `delegateType` is used for delegated events. These types are generally DOM event types, and *should not* be a special event themselves.

The behavior of these properties is easiest to see with an example. Assume a special event defined as follows:

```
1 | jQuery.event.special.pushy = {  
2 |     bindType: "click",  
3 |     delegateType: "click"  
4 | };
```

When these properties are defined, the following behavior occurs in the jQuery event system:

- Event handlers for the "pushy" event are actually attached to "click" -- both directly bound and delegated events.
- Special event hooks for "click" are called if they exist, *except* the `handle` hook for "pushy" is called when an event is delivered if one exists.

- Event handlers for "pushy" must be removed using the "pushy" event name, and are unaffected if "click" events are removed from the same elements.

So given the special event above, this code shows that a pushy isn't removed by removing clicks. That might be an effective way to defend against an ill-behaved plugin that didn't namespace its removal of click events, for example:

```
1  var $p = $("p");
2
3  $p.on( "click", function( e ) {
4      $("body").append( "I am a " + e.type + "!"
5  });
6
7  $p.on( "pushy", function( e ) {
8      $("body").append( "I am pushy but still a
9  });
10
11 $p.trigger("click"); // triggers both handle
12
13 $p.off("click");
14
15 $p.trigger("click"); // still triggers "push
16
17 $p.off("pushy");
```

These two properties are often used in conjunction with a `handle` hook function; the hook might, for example, change the event name from "click" to "pushy" before calling event handlers. See below for an example.

### The `handleObj` object

Many of the special event hook functions below are passed a `handleObj` object that provides more information about the event, how it was attached, and its current state. This object and its

contents should be treated as read-only data, and only the properties below are documented for use by special event handlers. For the discussion below, assume an event is attached with this code:

```
1 | $(".dialog").on( "click.myPlugin", "button",  
2 |   mydata: 42  
3 | }, myHandler );
```

**type:** String : The type of event, such as "click". When special event mapping is used via `bindType` or `delegateType`, this will be the mapped type.

**origType:** String : The original type name (in this case, "click") regardless of whether it was mapped via `bindType` or `delegateType`. So when a "pushy" event is mapped to "click" its `origType` would be "pushy". See the examples in those special event properties above for more detail.

**namespace:** String : Namespace(s), if any, provided when the event was attached, such as "myPlugin". When multiple namespaces are given, they are separated by periods and sorted in ascending alphabetical order. If no namespaces are provided, this property is an empty string.

**selector:** String : For delegated events, this is the selector used to filter descendant elements and determine if the handler should be called. In the example it is "button". For directly bound events, this property is `null`.

**data:** Object : The data, if any, passed to jQuery during event binding, e.g., `{ myData: 42 }`. If the data argument was omitted or `undefined`, this property is `undefined` as well.

**handler:** function( event: jQuery.Event ) : Event handler function passed to jQuery during event binding; in the example it is a reference to `myHandler`. If `false` was passed during event binding, the handler refers to a single shared function that simply returns `false`.

## **setup: function( data: Object, namespaces, eventHandle: function )**

The setup hook is called the first time an event of a particular type is attached to an element; this provides the hook an opportunity to do processing that will apply to all events of this type on this element. The `this` keyword will be a reference to the element where the event is being attached and `eventHandle` is jQuery's event handler function. In most cases the `namespaces` argument should not be used, since it only represents the namespaces of the *first* event being attached; subsequent events may not have this same namespaces.

This hook can perform whatever processing it desires, including attaching its own event handlers to the element or to other elements and recording setup information on the element using the `jQuery.data()` method. If the setup hook wants jQuery to add a browser event (via `addEventListener` or `attachEvent`, depending on browser) it should return `false`. In all other cases, jQuery will not add the browser event, but will continue all its other bookkeeping for the event. This would be appropriate, for example, if the event was never fired by the browser but invoked by `.trigger()`. To attach the jQuery event handler in the setup hook, use the `eventHandle` argument.

## **teardown: function()**

The teardown hook is called when the final event of a particular type is removed from an element. The `this` keyword will be a reference to the element where the event is being cleaned up. This hook should return `false` if it wants jQuery to remove the event from the browser's event system (via `removeEventListener` or `detachEvent`). In most cases, the setup and teardown hooks should return the same value.

If the setup hook attached event handlers or added data to an element through a mechanism such as `jQuery.data()`, the teardown hook should reverse the process and remove them. jQuery will generally remove the data and events when an element is totally removed from the document, but failing to remove data or events on teardown will cause a memory leak if the element stays in the document.

### **add: function( handleObj )**

Each time an event handler is added to an element through an API such as `.on()`, jQuery calls this hook. The `this` keyword will be the element to which the event handler is being added, and the `handleObj` argument is as described in the section above. The return value of this hook is ignored.

### **remove: function( handleObj )**

When an event handler is removed from an element using an API such as `.off()`, this hook is called. The `this` keyword will be the element where the handler is being removed, and the `handleObj` argument is as described in the section above. The return value of this hook is ignored.

### **trigger: function( event: jQuery.Event, data: Object )**

Called when the `.trigger()` or `.triggerHandler()` methods are used to trigger an event for the special type from code, as opposed to events that originate from within the browser. The `this` keyword will be the element being triggered, and the event argument will be a `jQuery.Event` object constructed from the caller's input. At minimum, the event type, data, namespace, and target properties are set on the event. The data argument represents additional data passed by `.trigger()` if present.

The trigger hook is called early in the process of triggering an event, just after the `jQuery.Event` object is constructed and before any handlers have been called. It can process the triggered event in any way, for example by calling `event.stopPropagation()` or `event.preventDefault()` before returning. If the hook returns `false`, jQuery does not perform any further event triggering actions and returns immediately. Otherwise, it performs the normal trigger processing, calling any event handlers for the element and bubbling the event (unless propagation is stopped in advance or `noBubble` was specified for the special event) to call event handlers attached to parent elements.

### **\_default: function( event: jQuery.Event, data: Object )**

When the `.trigger()` method finishes running all the event

handlers for an event, it also looks for and runs any method on the target object by the same name unless of the handlers called `event.preventDefault()`. So, `.trigger("submit")` will execute the `submit()` method on the element if one exists. When a `_default` hook is specified, the hook is called just prior to checking for and executing the element's default method. If this hook returns the value `false` the element's default method will be called; otherwise it is not.

### **handle: function( event: jQuery.Event, data: Object )**

jQuery calls a handle hook when the event has occurred and jQuery would normally call the user's event handler specified by `.on()` or another event binding method. If the hook exists, jQuery calls it *instead of* that event handler, passing it the event and any data passed from `.trigger()` if it was not a native event. The `this` keyword is the DOM element being handled, and `event.handleObj` property has the detailed event information.

Based in the information it has, the handle hook should decide whether to call the original handler function which is in `event.handleObj.handler`. It can modify information in the event object before calling the original handler, but *must restore* that data before returning or subsequent unrelated event handlers may act unpredictably. In most cases, the handle hook should return the result of the original handler, but that is at the discretion of the hook. The handle hook is unique in that it is the only special event function hook that is called under its original special event name when the type is mapped using `bindType` and `delegateType`. For that reason, it is almost always an error to have anything other than a handle hook present if the special event defines a `bindType` and `delegateType`, since those other hooks will never be called.

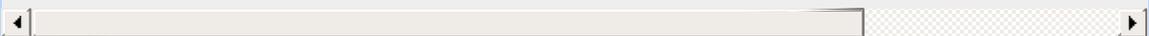
### **Example: Multiclick event**

This `multiclick` special event maps itself into a standard click event, but uses a handle hook so that it can monitor the event and only deliver it when the user clicks on the element a multiple of the number of times specified during event binding.

The hook stores the current click count in the data object, so multiclick handlers on different elements don't interfere with each

other. It changes the event type to the original "multiclick" type before calling the handler and restores it to the mapped "click" type before returning:

```
1  jQuery.event.special.multiclick = {
2    delegateType: "click",
3    bindType: "click",
4    handle: function( event ) {
5      var handleObj = event.handleObj;
6      var targetData = jQuery.data( event.target, "multiclick" );
7      var ret = null;
8
9      // If a multiple of the click count, run the handler
10     targetData.clicks = ( targetData.clicks || 0 ) + 1;
11
12     if ( targetData.clicks % event.data === 0 ) {
13       event.type = handleObj.origType;
14       ret = handleObj.handler.apply( this, arguments );
15       event.type = handleObj.type;
16       return ret;
17     }
18   }
19 };
20
21 // Sample usage
22 $("p").on( "multiclick", {
23   clicks: 3
24 }, function( event ) {
25   alert("clicked 3 times");
26 });
```



# Effects

jQuery makes it trivial to add simple effects to your page. Effects can use the built-in settings, or provide a customized duration. You can also create custom animations of arbitrary CSS properties.

For complete details on jQuery effects, visit the [Effects documentation on api.jquery.com](#).

- [Introduction to Effects](#)
- [Custom Effects with \\$.fn.animate](#)
- [Queue & Dequeue Explained](#)
- [The uses of jQuery .queue\(\) and .dequeue\(\)](#)

# Introduction to Effects

## Showing and Hiding Content

jQuery can show or hide content instantaneously with `$.fn.show` or `$.fn.hide`:

```
1 // Instantaneously hide all paragraphs
2 $("p").hide();
3
4 // Instantaneously show all divs that have the hidden class
5 $("div.hidden").show();
```

When jQuery hides an element, it sets its CSS `display` property to `none`. This means the content will have zero width and height; it does not mean that the content will simply become transparent and leave an empty area on the page.

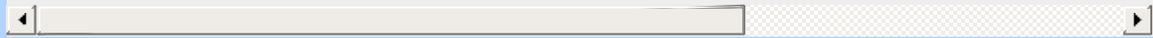
jQuery can also show or hide content by means of animation effects. You can tell `$.fn.show` and `$.fn.hide` to use animation in a couple of ways. One is to pass in a string-valued argument of `slow`, `normal`, or `fast`:

```
1 // Slowly hide all paragraphs
2 $("p").hide("slow");
3
4 // Quickly show all divs that have the hidden class
5 $("div.hidden").show("fast");
```

If you prefer more direct control over the duration of the animation effect, you can pass the desired duration in milliseconds to `$.fn.show` and `$.fn.hide`:

```
1 // Hide all paragraphs over half a second
2 $("p").hide( 500 );
```

```
3  
4 // Show all divs that have the hidden style class over  
5 $("div.hidden").show( 1250 );
```



Most developers pass in a number of milliseconds to have more precise control over the duration.

## Fade and Slide Animations

You may have noticed that `$.fn.show` and `$.fn.hide` use a combination of slide and fade effects when showing and hiding content in an animated way. If you would rather show or hide content with one effect or the other, there are additional methods that can help. `$.fn.slideDown` and `$.fn.slideUp` show and hide content, respectively, using only a slide effect. Slide animations are accomplished by rapidly making changes to an element's CSS `height` property.

```
1 // Hide all paragraphs using a slide up anima
2 $("p").slideUp( 800 );
3
4 // Show all hidden divs using a slide down an
5 $("div.hidden").slideDown( 600 );
```

Similarly `$.fn.fadeIn` and `$.fn.fadeOut` show and hide content, respectively, by means of a fade animation. Fade animations involve rapidly making changes to an element's CSS `opacity` property.

```
1 // Hide all paragraphs using a fade out anima
2 $("p").fadeOut( 1500 );
3
4 // Show all hidden divs using a fade in anima
5 $("div.hidden").fadeIn( 750 );
```

## Changing Display Based on Current Visibility State

jQuery can also let you change a content's visibility based on its current visibility state. `$.fn.toggle` will show content that is currently hidden and hide content that is currently visible. You can pass the same arguments to `$.fn.toggle` as you pass to any of the effects methods above.

```
1 // Instantaneously toggle the display of all
2 $("p").toggle();
3
4 // Slowly toggle the display of all images
5 $("img").toggle("slow");
6
7 // Toggle the display of all divs over 1.8 se
8 $("div").toggle( 1800 );
```

`$.fn.toggle` will use a combination of slide and fade effects, just as `$.fn.show` and `$.fn.hide` do. You can toggle the display of content with just a slide or a fade using `$.fn.slideToggle` and `$.fn.fadeToggle`.

```
1 // Toggle the display of all ordered lists ov
2 $("ol").slideToggle( 1000 );
3
4 // Toggle the display of all blockquotes over
5 $("blockquote").fadeToggle( 400 );
```

## Doing Something After an Animation Completes

A common mistake when implementing jQuery effects is assuming that the execution of the next method in your chain will wait until the animation runs to completion.

```
1 // Fade in all hidden paragraphs; then add a
2 $("p.hidden").fadeIn( 750 ).addClass("lookAtMe")
```

It is important to realize that `$.fn.fadeIn` above only *kicks off* the animation. Once started, the animation is implemented by rapidly changing CSS properties in a JavaScript `setInterval()` loop. When you call `$.fn.fadeIn`, it starts the animation loop and then returns the jQuery object, passing it along to `$.fn.addClass` which will then add the `lookAtMe` style class while the animation loop is just getting started.

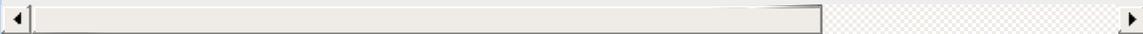
To defer an action until after an animation has run to completion, you need to use an animation callback function. You can specify your animation callback as the second argument passed to any of the animation methods discussed above. For the code snippet above, we can implement a callback as follows:

```
1 // Fade in all hidden paragraphs; then add a
2 $("p.hidden").fadeIn( 750, function(){
3     // this = DOM element which has just finished
4     $( this ).addClass("lookAtMe");
5 });
```

Note that you can use the keyword `this` to refer to the DOM element being animated. Also note that the callback will be called for each element in the jQuery object. This means that if your selector returns no elements, your animation callback will never

run! You can solve this problem by testing whether your selection returned any elements; if not, you can just run the callback immediately.

```
1 // Run a callback even if there were no elements
2 var $someElement = $("#nonexistent");
3
4 var cb = function() {
5     console.log("done!");
6 };
7
8 if ( $someElement.length ) {
9     $someElement.fadeIn( 300, cb );
10 } else {
11     cb();
12 }
```



## Managing Animation Effects

jQuery provides some additional features for controlling your animations:

`$.fn.stop`

`$.fn.stop` will immediately terminate all animations running on the elements in your selection. You might give end-users control over page animations by rigging a button they can click to stop the animations.

```
1 // Create a button to stop all animations on
2 $("input").attr({
3   type: "button",
4   value: "Stop All Animations"
5 }).on( "click", function() {
6   $("body *").filter(":animated").stop();
7 }).appendTo( document.body );
```

`$.fn.delay`

`$.fn.delay` is used to introduce a delay between successive animations. For example:

```
1 // Hide all level 1 headings over half a second
2 // and reveal all level 1 headings over 0.3 s
3 $("h1").hide( 500 ).delay( 1500 ).show( 300 )
```

`jQuery.fx`

The `jQuery.fx` object has a number of properties that control how effects are implemented. `jQuery.fx.speeds` maps the `slow`, `normal`, and `fast` duration arguments mentioned above to a specific number of milliseconds. The default value of `jQuery.fx.speeds` is:

```
1  {
2    slow: 600,
3    fast: 200,
4    // Default speed, used for "normal"
5    _default: 400
6  }
```

You can modify any of these settings and even introduce some of your own:

```
1  jQuery.fx.speeds.fast = 300;
2  jQuery.fx.speeds.blazing = 100;
3  jQuery.fx.speeds.excruciating = 60000;
```

`jQuery.fx.interval` controls the number of frames per second that are displayed in an animation. The default value is 13 milliseconds between successive frames. You can set this a lower value for faster browsers to make the animations run smoother. However this will mean more frames per second and thus a higher computational load for the browser, so you should be sure to test the performance implications of doing so thoroughly.

Finally, `jQuery.fx.off` can be set to true to disable all animations. Elements will immediately be set to the target final state instead. This can be especially useful when dealing with older browsers; you also may want to provide the option to disable all animations to your users.

```
1  $("input").attr({
2    type : "button",
3    value : "Disable Animations"
4  }).on( "click", function(){
5    jQuery.fx.off = true;
6  }).appendTo( document.body );
```



# Custom Effects with \$.fn.animate

jQuery makes it possible to animate arbitrary CSS properties via the `$.fn.animate` method. The `$.fn.animate` method lets you animate to a set value, or to a value relative to the current value.

```
1 // Custom effects with `$.fn.animate`">
2 $("div.funtimes").animate({
3     left : "+=50",
4     opacity : 0.25
5 },
6 // duration:
7 300,
8 // callback:
9 function() {
10     console.log("done!");
11 }
12 });
```

Color-related properties cannot be animated with `$.fn.animate` using jQuery out of the box. Color animations can easily be accomplished by including the [color plugin](#). We'll discuss using plugins later in the book.

## Easing

Definition: Easing describes the manner in which an effect occurs — whether the rate of change is steady, or varies over the duration of the animation. jQuery includes only two methods of easing: swing and linear. If you want more natural transitions in your animations, various easing plugins are available.

As of jQuery 1.4, it is possible to do per-property easing when using the `$.fn.animate` method.

```
1 // Per-property easing">
2 $("div.funtimes").animate({
3   left: [ "+=50", "swing" ],
4   opacity: [ 0.25, "linear" ]
5 }, 300 );
```

For more details on easing options, see [Animation documentation on api.jquery.com](#).

# Queue & Dequeue Explained

When you use the `animate` and `show`, `hide`, `slideUp`, etc effect methods, you're adding a job on to the fx queue. By default, using `queue` and passing a function, will add to the fx queue. So we're creating our own bespoke animation step:

```
1  $(".box").animate({
2    height : 20
3  }, "slow" ).queue(function() {
4
5    $("#title").html("We're in the animation, k
6
7  });
```

As I said though, these methods come in pairs, so anything you add using `queue`, you need to `dequeue` to allow the process to continue. In the code above, if I chained more animations on, until I call `$(this).dequeue()`, the subsequent animations wouldn't run:

```
1  $(".box").animate({
2    height : 20
3  }, "slow" ).queue(function() {
4
5    $("#title").html("We're in the animation,
6
7    $( this ).dequeue();
8
9  }).animate({
10   height: 150
11  });
```

Keeping in mind that the animation won't continue until we've explicitly called `dequeue`, we can easily create a pausing plugin, by adding a step in the queue that sets a timer and triggers after `n` milliseconds, at which time, it dequeues the element:

```
1  $.fn.pause = function( n ) {
2
3      return this.queue(function() {
4
5          var el = this;
6          setTimeout( function() {
7              return $( el ).dequeue();
8          }, n );
9      });
10
11 };
12
13 $(".box").animate({
14     height : 20
15 }, "slow" ).pause( 1000 ).animate({
16     height: 150
17 });
```

Remember that the first argument for `queue` and `dequeue` are `fx`, and that in all of these examples I'm not including it because jQuery set the argument to `fx` by default - so I don't have to specify it.

# The uses of jQuery .queue() and .dequeue()

Queues in jQuery are used for animations. You can use them for any purpose you like. They are an array of functions stored on a per element basis, using `jQuery.data()`. They are First-In-First-Out (FIFO). You can add a function to the queue by calling `.queue()`, and you remove (by calling) the functions using `.dequeue()`.

To understand the internal jQuery queue functions, reading the source and looking at examples helps me out tremendously. One of the best examples of a queue function I've seen is `.delay()`:

```
1  $.fn.delay = function( time, type ) {
2
3      time = jQuery.fx ? jQuery.fx.speeds[ time
4
5      type = type || "fx";
6
7      return this.queue( type, function() {
8
9          var elem = this;
10
11         setTimeout(function() {
12
13             jQuery.dequeue( elem, type );
14
15         }, time );
16
17     });
18
19 }
```

## The default queue – fx

The default queue in jQuery is fx. The default queue has some special properties that are not shared with other queues.

- Auto Start: When calling `$(elem).queue(function() {});` the fx queue will automatically dequeue the next function and run it if the queue hasn't started.
- 'inprogress' sentinel: Whenever you `dequeue()` a function from the fx queue, it will `unshift()` (push into the first location of the array) the string "inprogress" – which flags that the queue is currently being run.
- It's the default! The fx queue is used by `.animate()` and all functions that call it by default.

If you are using a custom queue, you must manually `.dequeue()` the functions, they will not auto start!

## Retrieving/Setting the queue

You can retrieve a reference to a jQuery queue by calling `.queue()` without a function argument. You can use the method if you want to see how many items are in the queue. You can use `push`, `pop`, `unshift`, `shift` to manipulate the queue in place. You can replace the entire queue by passing an array to the `.queue()` function.

## Quick Examples:

```
1 // lets assume $elem is a jQuery object that
2 var queue = $elem.queue();
3
4 // remove the last function from the animati
5 var lastFunc = queue.pop();
6
7 // insert it at the beginning:
8 queue.unshift( lastFunc );
9
10 // replace queue with the first three items
11 $elem.queue( queue.slice( 0, 3 ) );
```



## An animation (fx) queue example:

```
1 $(function() {
2
3 // lets do something with google maps:
4 var $map = $("#map_canvas");
5
6 var myLatLng = new google.maps.LatLng( -34
7
8 var myOptions = {
9   zoom: 8,
10  center: myLatLng,
11  mapTypeId: google.maps.MapTypeId.ROADMAP
12 };
13
14 var geocoder = new google.maps.Geocoder();
15
16 var map = new google.maps.Map( $map[0], my
17
```

```
18     var resized = function() {
19
20         // simple animation callback - let maps know we re
21         google.maps.event.trigger( map, "resize" );
22
23     };
24
25     // wait 2 seconds
26     $map.delay( 2000 );
27
28     // resize the div:
29     $map.animate({
30         width: 250,
31         height: 250,
32         marginLeft: 250,
33         marginTop:250
34     }, resized );
35
36     // geocode something
37     $map.queue(function( next ) {
38
39         // find stackoverflow's whois address:
40         geocoder.geocode( {
41             address: "55 Broadway New York NY 10006"
42         }, handleResponse );
43
44         function handleResponse( results, status )
45
46             if ( status === google.maps.GeocoderStatus.OK )
47
48                 var location = results[ 0 ].geometry.location
49
50                 map.setZoom( 13 );
51
52                 map.setCenter( location );
53
54                 new google.maps.Marker({
```

```
55         map: map,
56         position: location
57     });
58
59     }
60
61     // geocoder result returned, continue with anima
62     next();
63
64     }
65
66     });
67
68     // after we find stack overflow, wait 3 more seconds
69     $map.delay( 3000 );
70
71     // and resize the map again
72     $map.animate({
73         width: 500,
74         height: 500,
75         marginLeft:0,
76         marginTop: 0
77     }, resized );
78 });
```

## Queueing something like Ajax Calls:

```
1 // jQuery on an empty object, we are going t
2 var ajaxQueue = $({});
3
4 $.ajaxQueue = function( ajaxOpts ) {
5
6     // hold the original complete function
7     var oldComplete = ajaxOpts.complete;
8
```

```
9 // queue our ajax request
10 ajaxQueue.queue(function( next ) {
11
12 // create a complete callback to fire the next event
13 ajaxOpts.complete = function() {
14
15 // fire the original complete if it was there
16 if ( oldComplete ) {
17
18 oldComplete.apply( this, arguments );
19
20 }
21
22 // run the next query in the queue
23 next();
24
25 };
26
27 // run the query
28 $.ajax( ajaxOpts );
29
30 });
31
32 };
33
34 // get each item we want to copy
35 $("#items li").each(function( idx ) {
36
37 // queue up an ajax request
38 $.ajaxQueue({
39 url: "/ajax_html_echo/",
40 data: {
41 html : "[" + idx + "]" + $( this ).html()
42 },
43 type: "POST",
44 success: function( data ) {
45
```

```
46     // Write to #output
47     $("#output").append( $("#<li>", {
48         html: data
49     }));
50
51     }
52     });
53
54 });
```

## Another custom queue example

```
1  var theQueue = $({}); // jQuery on an empty
2
3  $.each( [1,2,3], function( i, num ) {
4
5      // lets add some really simple functions t
6      theQueue.queue( "alerts", function(next) {
7
8          // show something, and if they hit "yes"
9          if ( confirm("index:" + i + " = " + num
10
11              next();
12
13          }
14
15      });
16
17  });
18
19  // create a button to run the queue:
20  $("#<button>", {
21      text: "Run Queue",
22      click: function() {
23
```

```
24     theQueue.dequeue("alerts");
25
26     }
27   }).appendTo("body");
28
29   // create a button to show the length:
30   $("<button>", {
31     text: "Show Length",
32     click: function() {
33
34       alert(theQueue.queue("alerts").length);
35
36     }
37   }).appendTo("body");
```



# Ajax

Traditionally webpages required reloading to update their content. For web-based email this meant that users had to manually reload their inbox to check and see if they had new mail. This had huge drawbacks: it was slow and it required user input. When the user reloaded their inbox, the server had to reconstruct the entire web page and resend all of the HTML, CSS, JavaScript, as well as the user's email. This was hugely inefficient. Ideally, the server should only have to send the user's new messages, not the entire page. By 2003 all the major browsers, solved this issue by adopting the XMLHttpRequest (XHR) object, allowing browsers to communicate with the server without requiring a page reload.

The XMLHttpRequest object is part of a technology called Ajax (Asynchronous JavaScript and XML). Using Ajax, data could then be passed between the browser and the server, using the XMLHttpRequest API, without having to reload the web page. With the widespread adoption of the XMLHttpRequest object it quickly became possible to build web applications like Google Maps, and Gmail that used XMLHttpRequest to get new map tiles, or new email without having to reload the entire page.

Ajax requests are triggered by JavaScript code; your code sends a request to a URL, and when it receives a response, a callback function can be triggered to handle the response. Because the request is asynchronous, the rest of your code continues to execute while the request is being processed, so it's imperative that a callback be used to handle the response.

Unfortunately, different browsers implement the Ajax API differently. Typically this meant that developers would have to account for all the different browsers to ensure that Ajax would work universally. Fortunately, jQuery provides Ajax support that abstracts away painful browser differences. It offers both a full-featured `$.ajax()` method, and simple convenience methods such as `$.get()`, `$.getScript()`, `$.getJSON()`, `$.post()`, and `$.load()`.

Most jQuery applications don't in fact use XML, despite the name "Ajax"; instead, they transport data as plain HTML or JSON

(JavaScript Object Notation).

In general, Ajax does not work across domains. For instance, a webpage loaded from example1.com is unable to make an Ajax request to example2.com as it would violate the same origin policy. As a work around, JSONP (JSON with Padding) uses `<script>` tags to load files containing arbitrary JavaScript content and JSON, from another domain. More recently browsers have implemented a technology called Cross-origin resource sharing (CORS), that allows Ajax requests to different domains.

- [Key Concepts](#)
- [jQuery's Ajax-Related Methods](#)
- [Ajax and Forms](#)
- [Working with JSONP](#)
- [Ajax Events](#)

# Key Concepts

Proper use of Ajax-related jQuery methods requires understanding some key concepts first.

## GET vs. Post

The two most common “methods” for sending a request to a server are GET and POST. It’s important to understand the proper application of each.

The GET method should be used for non-destructive operations — that is, operations where you are only “getting” data from the server, not changing data on the server. For example, a query to a search service might be a GET request. GET requests may be cached by the browser, which can lead to unpredictable behavior if you are not expecting it. GET requests generally send all of their data in a query string.

The POST method should be used for destructive operations — that is, operations where you are changing data on the server. For example, a user saving a blog post should be a POST request. POST requests are generally not cached by the browser; a query string can be part of the URL, but the data tends to be sent separately as post data.

## Data Types

jQuery generally requires some instruction as to the type of data you expect to get back from an Ajax request; in some cases the data type is specified by the method name, and in other cases it is provided as part of a configuration object. There are several options:

### **text**

For transporting simple strings

### **html**

For transporting blocks of HTML to be placed on the page

### **script**

For adding a new script to the page

### **json**

For transporting JSON-formatted data, which can include strings, arrays, and objects

## **Note**

As of jQuery 1.4, if the JSON data sent by your server isn't properly formatted, the request may fail silently. See <http://json.org/> for details on properly formatting JSON, but as a general rule, use built-in language methods for generating JSON on the server to avoid syntax issues.

### **jsonp**

For transporting JSON data from another domain

### **xml**

For transporting data in a custom XML schema

I am a strong proponent of using the JSON format in most cases, as it provides the most flexibility. It is especially useful for sending both HTML and data at the same time.

## **A is for Asynchronous**

The asynchronicity of Ajax catches many new jQuery users off guard. Because Ajax calls are asynchronous by default, the response is not immediately available. Responses can only be handled using a callback. So, for example, the following code will not work:

```
1 | var response;  
2 |
```

```
3 | $.get( "foo.php", function( r ) {
4 |
5 |     response = r;
6 |
7 | });
8 |
9 | console.log( response ); // undefined
```

Instead, we need to pass a callback function to our request; this callback will run when the request succeeds, at which point we can access the data that it returned, if any.

```
1 | $.get( "foo.php", function( response ) {
2 |     console.log( response ); // server response
3 | });
```

## Same-Origin Policy and JSONP

In general, Ajax requests are limited to the same protocol (http or https), the same port, and the same domain as the page making the request. This limitation does not apply to scripts that are loaded via jQuery's Ajax methods.

The other exception is requests targeted at a JSONP service on another domain. In the case of JSONP, the provider of the service has agreed to respond to your request with a script that can be loaded into the page using a `<script>` tag, thus avoiding the same-origin limitation; that script will include the data you requested, wrapped in a callback function you provide.

## Ajax and Firebug

Firebug (or the Webkit Inspector in Chrome or Safari) is an invaluable tool for working with Ajax requests. You can see Ajax requests as they happen in the Console tab of Firebug (and in the Resources > XHR panel of Webkit Inspector), and you can click on

a request to expand it and see details such as the request headers, response headers, response content, and more. If something isn't going as expected with an Ajax request, this is the first place to look to track down what's wrong.

# jQuery's Ajax-Related Methods

While jQuery does offer many Ajax-related convenience methods, the core `$.ajax` method is at the heart of all of them, and understanding it is imperative. We'll review it first, and then touch briefly on the convenience methods.

I generally use the `$.ajax` method and do not use convenience methods. As you'll see, it offers features that the convenience methods do not, and its syntax is more easily understandable, in my opinion.

`$.ajax`

jQuery's core `$.ajax` method is a powerful and straightforward way of creating Ajax requests. It takes a configuration object that contains all the instructions jQuery requires to complete the request. The `$.ajax` method is particularly valuable because it offers the ability to specify both success and failure callbacks. Also, its ability to take a configuration object that can be defined separately makes it easier to write reusable code. For complete documentation of the configuration options, visit <http://api.jquery.com/jquery.ajax/>.

```
1 // Using the core $.ajax method
2 $.ajax({
3
4     // the URL for the request
5     url : "post.php",
6
7     // the data to send
8     // (will be converted to a query string)
9     data : {
10        id : 123
11    },
12
13    // whether this is a POST or GET request
14    type : "GET",
```

```

15
16 // the type of data we expect back
17 dataType : "json",
18
19 // code to run if the request succeeds;
20 // the response is passed to the function
21 success : function( json ) {
22     $("<h1/>").text( json.title ).appendTo("body"
23     $("<div class=\"content\"/>").html( json.html ).ap
24 },
25
26 // code to run if the request fails;
27 // the raw request and status codes are
28 // passed to the function
29 error : function( xhr, status ) {
30     alert("Sorry, there was a problem!");
31 },
32
33 // code to run regardless of success or failure
34 complete : function( xhr, status ) {
35     alert("The request is complete!");
36 }
37
38 });

```

### Note A note about the dataType setting: if the server sends back data that is in a different format than you specify, your code may fail, and the reason will not always be clear, because the HTTP response code will not show an error. When working with Ajax requests, make sure your server is sending back the data type you're asking for, and verify that the Content-type header is accurate for the data type. For example, for JSON data, the Content-type header should be `application/json`.

## `$.ajax` Options

There are many, many options for the `$.ajax` method, which is part

of its power. For a complete list of options, visit <http://api.jquery.com/jquery.ajax/>; here are several that you will use frequently:

### **async**

Set to `false` if the request should be sent synchronously. Defaults to `true`. Note that if you set this option to `false`, your request will block execution of other code until the response is received.

### **cache**

Whether to use a cached response if available. Defaults to true for all dataTypes except "script" and "jsonp". When set to false, the URL will simply have a cachebusting parameter appended to it.

### **complete**

A callback function to run when the request is complete, regardless of success or failure. The function receives the raw request object and the text status of the request.

### **context**

The scope in which the callback function(s) should run (i.e. what `this` will mean inside the callback function(s)). By default, `this` inside the callback function(s) refers to the object originally passed to `$.ajax`.

### **data**

The data to be sent to the server. This can either be an object or a query string, such as `foo=bar&baz=bim`.

### **dataType**

The type of data you expect back from the server. By default, jQuery will look at the MIME type of the response if no dataType is specified.

### **error**

A callback function to run if the request results in an error. The

function receives the raw request object and the text status of the request.

### **jsonp**

The callback name to send in a query string when making a JSONP request. Defaults to "callback".

### **success**

A callback function to run if the request succeeds. The function receives the response data (converted to a JavaScript object if the `dataType` was JSON), as well as the text status of the request and the raw request object.

### **timeout**

The time in milliseconds to wait before considering the request a failure.

### **traditional**

Set to true to use the param serialization style in use prior to jQuery 1.4. For details, see <http://api.jquery.com/jquery.param/>.

### **type**

The type of the request, "POST" or "GET". Defaults to "GET". Other request types, such as "PUT" and "DELETE" can be used, but they may not be supported by all browsers.

### **url**

The URL for the request.

The `url` option is the only required property of the `$.ajax` configuration object; all other properties are optional. This can also be passed as the first argument to `$.ajax`, and the options object as the second argument.

## **Convenience Methods**

If you don't need the extensive configurability of `$.ajax`, and you

don't care about handling errors, the Ajax convenience functions provided by jQuery can be useful, terse ways to accomplish Ajax requests. These methods are just "wrappers" around the core `$.ajax` method, and simply pre-set some of the options on the `$.ajax` method.

The convenience methods provided by jQuery are:

### **\$.get**

Perform a GET request to the provided URL.

### **\$.post**

Perform a POST request to the provided URL.

### **\$.getScript**

Add a script to the page.

### **\$.getJSON**

Perform a GET request, and expect JSON to be returned.

In each case, the methods take the following arguments, in order:

#### **url**

The URL for the request. Required.

#### **data**

The data to be sent to the server. Optional. This can either be an object or a query string, such as `foo=bar&baz=bim`.

### Note This option is not valid for `$.getScript`.

#### **success callback**

A callback function to run if the request succeeds. Optional. The function receives the response data (converted to a JavaScript object if the data type was JSON), as well as the text status of the request and the raw request object.

#### **data type**

The type of data you expect back from the server. Optional.

### Note This option is only applicable for methods that don't already specify the data type in their name.

```
1 // Using jQuery's Ajax convenience methods
2 // get plain text or html
3 $.get( "/users.php", {
4     userId : 1234
5 }, function( resp ) {
6     console.log( resp ); // server response
7 });
8
9 // add a script to the page, then run a func
10 $.getScript( "/static/js/myScript.js", funct
11
12     functionFromMyScript();
13
14 });
15
16 // get JSON-formatted data from the server
17 $.getJSON( "/details.php", function( resp )
18
19     // log each key in the response data
20     $.each( resp, function( key, value ) {
21         console.log( key + " : " + value );
22     });
23
24 });
```

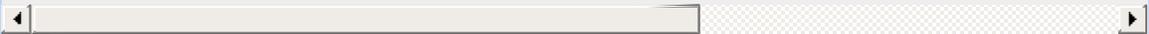
\$.fn.load

The `$.fn.load` method is unique among jQuery's Ajax methods in that it is called on a selection. The `$.fn.load` method fetches HTML from a URL, and uses the returned HTML to populate the selected element(s). In addition to providing a URL to the method, you can optionally provide a selector; jQuery will fetch only the matching

content from the returned HTML.

```
1 // Using $.fn.load to populate an element
2 $("#newContent").load("/foo.html");
```

```
1 // Using $.fn.load to populate an element bas
2 $("#newContent").load( "/foo.html #myDiv h1:f
3     alert("Content updated!");
4 });
```



# Ajax and Forms

jQuery's ajax capabilities can be especially useful when dealing with forms. There are several advantages, which can range from serialization, to simple client-side validation (e.g. "Sorry, that username is taken"), to [prefilters](#) (explained below), and even more!

## Serialization

Serializing form inputs in jQuery is extremely easy. Two methods come supported natively - `$.fn.serialize` and `$.fn.serializeArray`. While the names are fairly self-explanatory, there are many advantages to using them.

The `serialize` method serializes a form's data into a query string. For the element's value to be serialized, it **must** have a `name` attribute. Please note that values from inputs with a type of `checkbox` or `radio` are included only if they are checked.

```
1 // Turning form data into a query string
2 $("#myForm").serialize();
3
4 // creates a query string like this:
5 // field_1=something&field2=somethingElse
```

While plain old serialization is great, sometimes your application would work better if you sent over an array of objects, instead of just the query string. For that, jQuery has the `serializeArray` method. It's very similar to the `serialize` method listed above, except it produces an array of objects, instead of a string.

```
1 // Creating an array of objects containing f
2 $("#myForm").serializeArray();
3
4 // creates a structure like this:
5 // [
```

```
6 // {
7 //
8 //     name : "field_1",
9 //     value : "something"
10 //
11 // },
12 // {
13 //
14 //     name : "field_2",
15 //     value : "somethingElse"
16 //
17 // }
18 // ]
```

## Client-side validation

Client-side validation is, much like many other things, extremely easy using jQuery. While there are several cases developers can test for, some of the most common ones are: presence of a required input, valid usernames/emails/phone numbers/etc..., or checking an "I agree..." box.

Please note that it is advisable that you also perform server-side validation for your inputs. However, it typically makes for a better user experience to be able to validate some things without submitting the form.

With that being said, let's jump on in to some examples! First, we'll see how easy it is to check if a required field doesn't have anything in it. If it doesn't, then we'll `return false`, and prevent the form from processing.

```
1 // Using validation to check for the presence
2 $("#form").submit(function( event ) {
3
4     // if .required's value's length is zero
5     if ( $(".required").val().length === 0 ) {
```

```
6
7     // usually show some kind of error message here
8
9     // this prevents the form from submitting
10    return false;
11
12    } else {
13
14        // run $.ajax here
15
16    }
17
18    });
```

Let's see how easy it is to check for invalid characters in a username:

```
1 // Validate a phone number field
2 $("#form").submit(function( event ) {
3
4     var inputtedPhoneNumber = $("#phone").val(
5     // match only numbers
6     var phoneNumberRegex = /^\d*$/;
7
8     // if the phone number doesn't match the r
9     if ( !phoneNumberRegex.test( inputtedPhone
10
11     // usually show some kind of error messa
12
13     // prevent the form from submitting
14     return false;
15
16 } else {
17
18     // run $.ajax here
```

```
19
20     }
21
22 });
```

## Prefiltering

A prefilter is a way to modify the ajax options before each request is sent (hence, the name `prefilter`).

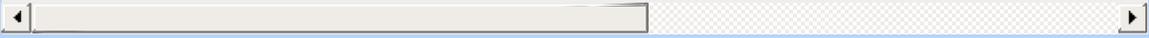
For example, say we would like to modify all `crossDomain` requests through a proxy. To do so with a prefilter is quite simple:

```
1 // Using a proxy with a prefilter
2 $.ajaxPrefilter(function( options, originalC
3
4     if ( options.crossDomain ) {
5
6         options.url = "http://mydomain.net/proxy
7
8         options.crossDomain = false;
9
10    }
11
12 });
```

You can pass in an optional argument before the callback function that specifies which `dataTypes` you'd like the prefilter to be applied to. For example, if we want our prefilter to only apply to `JSON` and `script` requests, we'd do:

```
1 // Using the optional dataType argument">
2 $.ajaxPrefilter( "json script", function( opt
3
```

```
4 // do all of the prefiltering here, but only for
5 // requests that indicate a dataType of "JSON" or "sc
6
7 });
```



# Working with JSONP

The advent of JSONP — essentially a consensual cross-site scripting hack — has opened the door to powerful mashups of content. Many prominent sites provide JSONP services, allowing you access to their content via a predefined API. A particularly great source of JSONP-formatted data is the [Yahoo! Query Language](#), which we'll use in the following example to fetch news about cats.

```
1 // Using YQL and JSONP
2 $.ajax({
3
4     url : "http://query.yahooapis.com/v1/publi
5
6     // the name of the callback parameter,
7     // as specified by the YQL service
8     jsonp : "callback",
9
10    // tell jQuery we're expecting JSONP
11    dataType : "jsonp",
12
13    // tell YQL what we want and that we want
14    data : {
15        q : "select title,abstract,url from sear
16        format : "json"
17    },
18
19    // work with the response
20    success : function( response ) {
21        console.log( response ); // server respo
22    }
23
24 });
```

jQuery handles all the complex aspects of JSONP behind-the-scenes — all we have to do is tell jQuery the name of the JSONP callback parameter specified by YQL ("callback" in this case), and otherwise the whole process looks and feels like a normal Ajax request.

# Ajax Events

Often, you'll want to perform an operation whenever an Ajax requests starts or stops, such as showing or hiding a loading indicator. Rather than defining this behavior inside every Ajax request, you can bind Ajax events to elements just like you'd bind other events. For a complete list of Ajax events, visit [Ajax Events documentation on docs.jquery.com](https://docs.jquery.com/Ajax/Events).

```
1 // Setting up a loading indicator using Ajax
2 $("#loading_indicator").ajaxStart(function()
3
4     $( this ).show();
5
6 }).ajaxStop(function() {
7
8     $( this ).hide();
9
10 });
```

# Plugins

A jQuery plugin is simply a new method that we use to extend jQuery's prototype object. By extending the prototype object you enable all jQuery objects to inherit any methods that you add. As established, whenever you call `jQuery()` you're creating a new jQuery object, with all of jQuery's methods inherited.

The idea of a plugin is to do something with a collection of elements. You could consider each method that comes with the jQuery core a plugin, like `fadeOut` or `addClass`.

You can make your own plugins and use them privately in your code or you can release them into the wild. There are thousands of jQuery plugins available online. The barrier to creating a plugin of your own is so low that you'll want to do it straight away!

- [Finding & Evaluating Plugins](#)
- [How to Create a Basic Plugin](#)
- [Advanced Plugin Concepts](#)
- [Writing Stateful Plugins with the jQuery UI Widget Factory](#)

# Finding & Evaluating Plugins

## Finding & Evaluating Plugins

One of the most celebrated aspects of jQuery is its extensive plugin ecosystem. From table sorting to form validation to autocompletion ... if there's a need for it, chances are good that someone has written a plugin for it.

The quality of jQuery plugins varies widely. Many plugins are extensively tested and well-maintained, but others are hastily created and then ignored. More than a few fail to follow best practices. Some plugins, mainly [jQuery UI](#), are maintained by the jQuery team. The quality of these plugins is as good as jQuery itself.

Google is your best initial resource for locating plugins, though the jQuery team is working on an improved plugin repository. Once you've identified some options via a Google search, you may want to consult the jQuery mailing list or the `#jquery` IRC channel to get input from others.

When looking for a plugin to fill a need, do your homework. Ensure that the plugin is well-documented, and look for the author to provide lots of examples of its use. Be wary of plugins that do far more than you need; they can end up adding substantial overhead to your page. For more tips on spotting a subpar plugin, read [Signs of a poorly written jQuery plugin](#) by Remy Sharp.

Once you choose a plugin, you'll need to add it to your page. Download the plugin, unzip it if necessary, place it your application's directory structure, then include the plugin in your page using a script tag (after you include jQuery).

# How to Create a Basic Plugin

Sometimes you want to make a piece of functionality available throughout your code; for example, perhaps you want a single method you can call on a jQuery selection that performs a series of operations on the selection. Maybe you wrote a really useful utility function that you want to be able to move easily to other projects. In this case, you may want to write a plugin.

## How jQuery Works 101: jQuery Object Methods and Utility Methods

Before we write our own plugins, we must first understand a little about how jQuery works. Take a look at this code:

```
1 | $("a").css( "color", "red" );
```

This is some pretty basic jQuery code, but do you know what's happening behind the scenes? Whenever you use the `$` function to select elements, it returns a jQuery object. This object contains all of the methods you've been using (`css()`, `click()`, etc.), and all of the elements that fit your selector. The jQuery object gets these methods from the `$.fn` object. This object contains all of the jQuery object methods, and if we want to write our own methods, it will need to contain those as well.

Additionally the jQuery utility method `$.trim()` is used above to remove any leading or trailing empty space characters from the user input. Utility methods are functions that reside directly on `$` function itself. You may occasionally want to write a utility method plugin when your extension to the jQuery API does not have to do something to a set of DOM elements you've retrieved.

## Basic Plugin Authoring

Let's say we want to create a plugin that makes text within a set of retrieved elements green. All we have to do is add a function called `greenify` to `$.fn` and it will be available just like any other jQuery object method.

```
1 | $.fn.greenify = function() {  
2 |     this.css( "color", "green" );  
3 | }  
4 |  
5 | $("a").greenify(); // makes all the links gr
```

Notice that to use `css()`, another method, we use `this`, not `$( this )`. This is because our `greenify` function is a part of the same object as `css()`.

## Chaining

This works, but there's a couple of things we need to do for our plugin to survive in the real world. One of jQuery's features is chaining, when you link five or six actions onto one selector. This is accomplished by having all jQuery object methods return the original jQuery object again (there are a few exceptions: `width()` called without parameters returns the width of the selected element, and is not chainable). Making our plugin method chainable takes one line of code:

```
1 | $.fn.greenify = function() {  
2 |     this.css( "color", "green" );  
3 |     return this;  
4 | }  
5 |  
6 | $("a").greenify().addClass("greenified");
```

Note that the notion of chaining is *not* applicable to jQuery utility methods like `$.trim()`.

## Protecting the \$ Alias and Adding Scope

The `$` variable is very popular among JavaScript libraries, and if you're using another library with jQuery, you will have to make jQuery not use the `$` with `jQuery.noConflict()`. However, this will break our plugin since it is written with the assumption that `$` is an alias to the `jQuery` function. To work well with other plugins, *and* still use the jQuery `$` alias, we need to put all of our code inside of an [Immediately Invoked Function Expression](#), and then pass the function `jQuery`, and name the parameter `$`:

```
1  (function ( $ ) {
2    $.fn.greenify = function() {
3      this.css( "color", "green" );
4      return this;
5    }
6
7    $.ltrim = function( str ) {
8      return str.replace(/^\s+/, '');
9    }
10   $.rtrim = function( str ) {
11     return str.replace(/\s+$/, '');
12   }
13
14 }( jQuery ));
```

In addition, the primary purpose of an Immediately Invoked Function is to allow us to have our own private variables. Pretend we want a different color green, and we want to store it in a variable.

```
1  (function ( $ ) {
2    var shade = "#556B2F";
3
4    $.fn.greenify = function() {
```

```
5     this.css( "color", shade );  
6     return this;  
7     }  
8  
9 }( jQuery ));
```

## Minimizing Plugin Footprint

It's good practice when writing plugins to only take up one slot within `$.fn`. This reduces both the chance that your plugin will be overridden, and the chance that your plugin will override other plugins. In other words, this is bad:

```
1  (function( $ ) {
2
3      $.fn.openPopup = function() {
4          // Open popup code
5      };
6
7      $.fn.closePopup = function() {
8          // Close popup code
9      };
10
11 }( jQuery ));
```

It would be much better to have one slot, and use parameters to control what action that one slot performs.

```
1  (function( $ ) {
2
3      $.fn.popup = function( action ) {
4
5          if ( action === "open" ) {
6
7              // Open popup code
8
9          } if ( action === "close" ) {
10
11              // Close popup code
12
13          }
14      }
15  });
```

```
14  
15     };  
16 }( jQuery ));
```

## Using the `each()` Method

Your typical jQuery object will contain references to any number of DOM elements, and that's why jQuery objects are often referred to as collections.

If you want to do any manipulating with specific elements (e.g. getting data an attribute, calculating specific positions) then you need to use `each()` to loop through the elements.

```
1  $.fn.myNewPlugin = function() {
2
3      return this.each(function() {
4
5          // do something to each element here
6
7      });
8
9  };
```

Notice that we return the results of `each()` instead of returning `this`. Since `each()` is already chainable, it returns `this`, which we then return. This is a better way to maintain chainability than what we've been doing so far.

## Accepting Options

As your plugins get more and more complex, it's a good idea to make your plugin customizable by accepting options. The easiest way to do this, especially if there are lots of options, is with an object literal. Let's change our greenify plugin to accept some options.

```
1  (function ( $ ) {
2
3      $.greenify = function( options ) {
4
5          // This is the easiest way to have default
6          var settings = $.extend({
7              "color": "#556B2F", // These are the
8              "background-color": "white"
9          }, options );
10
11         // Greenify the collection based on the
12         return this.css({
13             "color": settings.color,
14             "background-color": settings.background
15         });
16
17     };
18 }( jQuery ));
```

Example usage:

```
1  $("div").greenify({
2      "color": "orange"
3  });
```

The default value for `color` of `#556B2F` gets overridden by `$.extend()` to be orange.



## Putting It Together

Here's an example of a small plugin using some of the techniques we've discussed:

```
1  (function( $ ){
2    $.fn.showLinkLocation = function() {
3
4      return this.filter("a").each(function()
5
6        $( this ).append( " (" + $( this ).att
7
8      });
9
10   };
11
12   }( jQuery ));
13
14   // Usage example:
15   $("a").showLinkLocation();
```

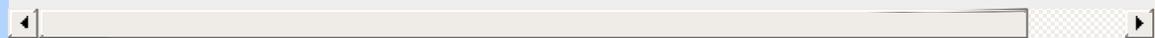
This handy plugin goes through all anchors in the collection and appends the href attribute in brackets.

```
1  <!-- Before plugin is called: -->
2  <a href="page.html">Foo</a>
3
4  <!-- After plugin is called: -->
5  <a href="page.html">Foo [page.html]</a>
```

Our plugin can be optimized though:

```
1  (function( $ ){
```

```
2
3     $.fn.showLinkLocation = function() {
4
5         return this.filter("a").append(function() {
6
7             return " (" + this.href + ")";
8
9         });
10    };
11
12 }( jQuery );
```



We're using the `append()` method's capability to accept a callback, and the return value of that callback will determine what is appended to each element in the collection. Notice also that we're not using the `attr` method to retrieve the href attribute, because the native DOM API gives us easy access with the aptly named href property.

# Advanced Plugin Concepts

## Provide public access to default plugin settings

An improvement we can, and should, make to the code above is to expose the default plugin settings. This is important because it makes it very easy for plugin users to override/customize the plugin with minimal code. And this is where we begin to take advantage of the function object.

```
1 // plugin definition
2 $.fn.hilight = function( options ) {
3
4     // Extend our default options with those p
5     // Note that the first arg to extend is an
6     // this is to keep from overriding our "de
7     var opts = $.extend({}, $.fn.hilight.defau
8
9     // Our plugin implementation code goes her
10
11 };
12
13 // plugin defaults - added as a property on
14 $.fn.hilight.defaults = {
15     foreground: "red",
16     background: "yellow"
17 };
```

Now users can include a line like this in their scripts:

```
1 // this need only be called once and does not
2 // have to be called from within a 'ready' bl
```

```
3 | $.fn.highlight.defaults.foreground = "blue";
```

And now we can call the plugin method like this and it will use a blue foreground color:

```
1 | $("#myDiv").highlight();
```

As you can see, we've allowed the user to write a single line of code to alter the default foreground color of the plugin. And users can still selectively override this new default value when they want:

```
1 | // override plugin default foreground color
2 | $.fn.highlight.defaults.foreground = "blue";
3 |
4 | // ...
5 |
6 | // invoke plugin using new defaults
7 | $(".highlightDiv").highlight();
8 |
9 | // ...
10 |
11 | // override default by passing options to pl
12 | $("#green").highlight({
13 |     foreground: "green"
14 | });
```

## Provide public access to secondary functions as applicable

This item goes hand-in-hand with the previous item and is an interesting way to extend your plugin (and to let others extend your plugin). For example, the implementation of our plugin may define a function called "format" which formats the highlight text. Our plugin

may now look like this, with the default implementation of the format method defined below the hilight function.

```
1 // plugin definition
2 $.fn.hilight = function( options ) {
3
4     // iterate and reformat each matched element
5     return this.each(function() {
6
7         var $this = $( this );
8
9         // ...
10
11        var markup = $this.html();
12
13        // call our format function
14        markup = $.fn.hilight.format( markup );
15
16        $this.html( markup );
17
18    });
19 };
20
21 // define our format function
22 $.fn.hilight.format = function( txt ) {"
23
24     return "<strong>" + txt + "</strong>";
25
26 };
```

We could have just as easily supported another property on the options object that allowed a callback function to be provided to override the default formatting. That's another excellent way to support customization of your plugin. The technique shown here takes this a step further by actually exposing the format function so that it can be redefined. With this technique it would be possible for

others to ship their own custom overrides of your plugin ?in other words, it means others can write plugins for your plugin.

Considering the trivial example plugin we're building in this article, you may be wondering when this would ever be useful. One real-world example is the [Cycle Plugin](#). The Cycle Plugin is a slideshow plugin which supports a number of built-in transition effects ?scroll, slide, fade, etc. But realistically, there is no way to define every single type of effect that one might wish to apply to a slide transition. And that's where this type of extensibility is useful. The Cycle Plugin exposes a "transitions" object to which users can add their own custom transition definitions. It's defined in the plugin like this:

```
1 | $.fn.cycle.transitions = {  
2 |  
3 |     // ...  
4 |  
5 | };
```

This technique makes it possible for others to define and ship transition definitions that plug-in to the Cycle Plugin.

## Keep private functions private

The technique of exposing part of your plugin to be overridden can be very powerful. But you need to think carefully about what parts of your implementation to expose. Once it's exposed, you need to keep in mind that any changes to the calling arguments or semantics may break backward compatibility. As a general rule, if you're not sure whether to expose a particular function, then you probably shouldn't.

So how then do we define more functions without cluttering the namespace and without exposing the implementation? This is a job for closures. To demonstrate, we'll add another function to our plugin called "debug". The debug function will log the number of selected elements to the Firebug console. To create a closure, we wrap the entire plugin definition in a function (as detailed in the

jQuery Authoring Guidelines).

```
1 // create closure
2 (function( $ ) {
3
4 // plugin definition
5 $.fn.highlight = function( options ) {
6     debug( this );
7     // ...
8 };
9
10 // private function for debugging
11 function debug( $obj ) {
12     if ( window.console && window.console.log )
13         window.console.log( "highlight selector" );
14 }
15 };
16
17 // ...
18
19 // end of closure
20 })( jQuery );
```

Our "debug" method cannot be accessed from outside of the closure and thus is private to our implementation.

## Support the Metadata Plugin

Depending on the type of plugin you're writing, adding support for the [Metadata Plugin](#) can make it even more powerful. Personally, I love the Metadata Plugin because it lets you use unobtrusive markup to override plugin options (which is particularly useful when creating demos and examples). And supporting it is very simple!

```
1 // plugin definition
```

```

2   $.fn.highlight = function( options ) {
3
4   // build main options before element iteration
5   var opts = $.extend( {}, $.fn.highlight.defaults, opti
6
7   return this.each(function() {
8       var $this = $( this );
9
10  // build element specific options
11  // This changed line tests to see if the Metadata
12  // and if it is, it extends our options object with
13  var o = $.meta ? $.extend( {}, opts, $this
14
15  //...
16  });
17  };

```

This line is added as the last argument to *jQuery.extend* so it will override any other option settings. Now we can drive behavior from the markup if we choose:

```

1   <!-- markup -->
2   <div class="highlight { background: 'red', for
3   Have a nice day!
4   </div>
5   <div class="highlight { foreground: 'orange' }
6   Have a nice day!
7   </div>
8   <div class="highlight { background: 'green' }"
9   Have a nice day!
10  </div>

```

And now we can highlight each of these divs uniquely using a single line of script:

```
1 | $(".highlight").highlight();
```

## Bob and Sue

Let's say Bob has created a wicked new gallery plugin (called "superGallery") which takes a list of images and makes them navigable. Bob's thrown in some animation to make it more interesting. He's tried to make the plugin as customizable as possible, and has ended up with something like this:

```
1 | jQuery.fn.superGallery = function( options )
2 |
3 |     // Bob's default settings:
4 |     var defaults = {
5 |         textColor : "#000",
6 |         backgroundColor : "#FFF",
7 |         fontSize : "1em",
8 |         delay : "quite long",
9 |         getTextFromTitle : true,
10 |        getTextFromRel : false,
11 |        getTextFromAlt : false,
12 |        animateWidth : true,
13 |        animateOpacity : true,
14 |        animateHeight : true,
15 |        animationDuration : 500,
16 |        clickImgToGoToNext : true,
17 |        clickImgToGoToLast : false,
18 |        nextButtonText : "next",
19 |        previousButtonText : "previous",
20 |        nextButtonTextcolor : "red",
21 |        previousButtonTextcolor : "red"
22 |    };
23 |
24 |     var settings = $.extend( {}, defaults, opt
25 |
```

```
26     return this.each(function() {
27         // Plugin code would go here...
28     });
29 };
```

The first thing that probably comes to your mind (ok, maybe not the first) is the prospect of how huge this plugin must be to accommodate such a level of customization. The plugin, if it weren't fictional, would probably be a lot larger than necessary. There are only so many kilobytes people will be willing to spend!

Now, our friend Bob thinks this is all fine; in fact, he's quite impressed with the plugin and its level of customization. He believes that all the options make for a more versatile solution, one which can be used in many different situations.

Sue, another friend of ours, has decided to use this new plugin. She has set up all of the options required and now has a working solution sitting in front of her. It's only five minutes later, after playing with the plugin, that she realizes the gallery would look much nicer if each image's width were animated at a slower speed. She hastily searches through Bob's documentation but finds no *animateWidthDuration* option!

## Do you see the problem?

It's not really about how many options your plugin has; but what options it has!

Bob has gone a little over the top. The level of customization he's offering, while it may seem high, is actually quite low, especially considering all the possible things one might want to control when using this plugin. Bob has made the mistake of offering a lot of ridiculously specific options, rendering his plugin much more difficult to customize!

## A better model

So it's pretty obvious: Bob needs a new customization model, one

which does not relinquish control or abstract away the necessary details.

The reason Bob is so drawn to this high-level simplicity is that the jQuery framework very much lends itself to this mindset. Offering a *previousButtonTextColor* option is nice and simple, but let's face it, the vast majority of plugin users are going to want way more control!

Here are a few tips which should help you create a better set of customizable options for your plugins:

## Don't create plugin-specific syntax

Developers who use your plugin shouldn't have to learn a new language or terminology just to get the job done.

Bob thought he was offering maximum customization with his *delay* option (look above). He made it so that with his plugin you can specify four different delays, "quite short," "very short," "quite long," or "very long":

```
1  var delayDuration = 0;
2
3  switch ( settings.delay ) {
4
5      case "very short":
6          delayDuration = 100;
7          break;
8
9      case "quite short":
10         delayDuration = 200;
11         break;
12
13         case "quite long":
14             delayDuration = 300;
15             break;
16
17         case "very long":
```

```
18     delayDuration = 400;
19     break;
20
21     default:
22         delayDuration = 200
23 }
```

Not only does this limit the level of control people have, but it takes up quite a bit of space. Twelve lines of code just to define the delaying time is a bit much, don't you think? A better way to construct this option would be to let plugin users specify the amount of time (in milliseconds) as a number, so that no processing of the option needs to take place.

The key here is not to diminish the level of control through your abstraction. Your abstraction, whatever it is, can be as simplistic as you want, but make sure that people who use your plugin will still have that much-sought-after low-level control! (By low-level I mean non-abstracted)

## Give full control of elements

If your plugin creates elements to be used within the DOM, then it's a good idea to offer plugin users some way to access those elements. Sometimes this means giving certain elements IDs or classes. But note that your plugin shouldn't rely on these hooks internally:

A bad implementation:

```
1 // Plugin code
2 $("
```

```
9     .appendTo( settings.container );
10
11     $wrapper.appendTo( settings.container ); // Easy to reference later...
```

Notice that we've created a reference to the injected wrapper and we're also calling the 'attr' method to add any specified attributes to the element. So, in our settings it might be handled like this:

```
1     var defaults = {
2         wrapperAttrs : {
3             id: "gallery-wrapper"
4         },
5         // ... rest of settings ...
6     };
7
8     // We can use the extend method to merge opt
9     // But with the added first parameter of TR
10    var settings = $.extend( true, {}, defaults,
```

The `$.extend()` method will now recurse through all nested objects to give us a merged version of both the defaults and the passed options, giving the passed options precedence.

The plugin user now has the power to specify any attribute of that wrapper element so if they require that there be a hook for any CSS styles then they can quite easily add a class or change the name of the ID without having to go digging around in plugin source.

The same model can be used to let the user define CSS styles:

```
1     var defaults = {
2         wrapperCSS : {},
3         // ... rest of settings ...
4     };
5
```

```
6 // Later on in the plugin where we define the wrapper:
7 var $wrapper = $( "<div />" )
8   .attr( settings.wrapperAttrs )
9   .css( settings.wrapperCSS ) // ** Set CSS!
10  .appendTo( settings.container );
```

Your plugin may have an associated StyleSheet where developers can add CSS styles. Even in this situation it's a good idea to offer some convenient way of setting styles in JavaScript, without having to use a selector to get at the elements.

## Provide callback capabilities

*What is a callback?* - A callback is essentially a function to be called later, normally triggered by an event. It's passed as an argument, usually to the initiating call of a component. (in this case, a jQuery plugin).

If your plugin is driven by events then it might be a good idea to provide a callback capability for each event. Plus, you can create your own custom events and then provide callbacks for those. In this gallery plugin it might make sense to add an 'onImageShow' callback.

```
1  var defaults = {
2    // we define an empty anonymous function
3    // so that we don't need to check its
4    // existence before calling it.
5    onImageShow : function() {},
6
7    // ... rest of settings ...
8
9  };
10
11 // Later on in the plugin:
12
13 $nextButton.bind( "click", showNextImage );
```

```
14
15  function showNextImage() {
16
17      // stuff to show the image here...
18
19      // Here's the callback:
20      settings.onImageShow.call( this );
21  }
```

Instead of initiating the callback via traditional means (adding parenthesis) we're calling it in the context of 'this' which will be a reference to the image node. This means that you have access to the actual image node through the 'this' keyword within the callback:

```
1  $("ul.imgs li").superGallery({
2      onImageShow : function() {
3          $( this ).after( "<span>" + $( this ).att
4      },
5
6      // ... other options ...
7  });
```

Similarly you could add an "onImageHide" callback and numerous other ones...

The point with callbacks is to give plugin users an easy way to add additional functionality without digging around in the source.

## Remember, it's a compromise

Your plugin is not going to be able to work in every situation. And equally, it's not going to be very useful if you offer no or very few methods of control. So, remember, it's always going to be a compromise. Three things you must always take into account are:

- *Flexibility*: How many situations will your plugin be able to deal with?
- *Size*: Does the size of your plugin correspond to its level of functionality? I.e. Would you use a very basic tooltip plugin if it was 20k in size? - Probably not!
- *Performance*: Does your plugin heavily process the options in any way? Does this effect speed? Is the overhead caused worth it for the end user?

# Writing Stateful Plugins with the jQuery UI Widget Factory

## Writing Stateful Plugins with the jQuery UI Widget Factory

While most existing jQuery plugins are stateless — that is, we call them on an element and that is the extent of our interaction with the plugin — there's a large set of functionality that doesn't fit into the basic plugin pattern.

In order to fill this gap, jQuery UI has implemented a more advanced plugin system. The new system manages state, allows multiple functions to be exposed via a single plugin, and provides various extension points. This system is called the widget factory and is exposed as `jQuery.widget` as part of jQuery UI 1.8; however, it can be used independently of jQuery UI.

To demonstrate the capabilities of the widget factory, we'll build a simple progress bar plugin.

To start, we'll create a progress bar that just lets us set the progress once. As we can see below, this is done by calling `jQuery.widget` with two parameters: the name of the plugin to create and an object literal containing functions to support our plugin. When our plugin gets called, it will create a new plugin instance and all functions will be executed within the context of that instance. This is different from a standard jQuery plugin in two important ways. First, the context is an object, not a DOM element. Second, the context is always a single object, never a collection.

A simple, stateful plugin using the jQuery UI widget factory

```
1 | $.widget( "nmk.progressbar", {
2 |
3 |   _create: function() {
4 |
5 |     var progress = this.options.value + "%";
6 |
7 |     this.element.addClass("progressbar").tex
8 |
9 |   }
```

```
10 |
11 | });
```

The name of the plugin must contain a namespace; in this case we've used the `nmk` namespace. There is a limitation that namespaces be exactly one level deep — that is, we can't use a namespace like `nmk.foo`. We can also see that the widget factory has provided two properties for us. `this.element` is a jQuery object containing exactly one element. If our plugin is called on a jQuery object containing multiple elements, a separate plugin instance will be created for each element, and each instance will have its own `this.element`. The second property, `this.options`, is a hash containing key/value pairs for all of our plugin's options. These options can be passed to our plugin as shown here.

### Note In our example we use the `nmk` namespace. The `ui` namespace is reserved for official jQuery UI plugins. When building your own plugins, you should create your own namespace. This makes it clear where the plugin came from and whether it is part of a larger collection.

Passing options to a widget

```
1 | $("<div></div>").appendTo( "body" ).progr
```

When we call `jQuery.widget` it extends jQuery by adding a method to `jQuery.fn` (the same way we'd create a standard plugin). The name of the function it adds is based on the name you pass to `jQuery.widget`, without the namespace; in our case it will create `jQuery.fn.progressbar`. The options passed to our plugin get set in `this.options` inside of our plugin instance. As shown below, we can specify default values for any of our options. When designing your API, you should figure out the most common use case for your plugin so that you can set appropriate default values and make all options truly optional.

```
1 | Setting default options for a widget
```

```

2
3 $.widget( "nmk.progressbar", {
4
5     // default options
6     options: {
7
8         value: 0
9
10    },
11
12    _create: function() {
13
14        var progress = this.options.value + "%";
15
16        this.element.addClass( "progressbar" ).text( progr
17
18    }
19
20 });

```

## Adding Methods to a Widget

Now that we can initialize our progress bar, we'll add the ability to perform actions by calling methods on our plugin instance. To define a plugin method, we just include the function in the object literal that we pass to `jQuery.widget`. We can also define "private" methods by prepending an underscore to the function name.

Creating widget methods

```

1 $.widget( "nmk.progressbar", {
2     options: {
3         value: 0
4     },
5
6     _create: function() {

```

```
7
8     var progress = this.options.value + "%";
9
10    this.element.addClass("progressbar").text( progress
11
12    },
13
14    // create a public method
15    value: function( value ) {
16
17        // no value passed, act as a getter
18        if ( value === undefined ) {
19
20            return this.options.value;
21
22            // value passed, act as a setter
23        } else {
24
25            this.options.value = this._constrain( value );
26
27            var progress = this.options.value + "%";
28
29            this.element.text( progress );
30
31        }
32
33    },
34
35    // create a private method
36    _constrain: function( value ) {
37
38        if ( value > 100 ) {
39
40            value = 100;
41
42        }
43
```

```
44     if ( value < 0 ) {
45         value = 0;
46     }
47
48     return value;
49
50 }
51
52 }
53
54 });
```

To call a method on a plugin instance, you pass the name of the method to the jQuery plugin. If you are calling a method that accepts parameters, you simply pass those parameters after the method name.

### Calling methods on a plugin instance

```
1  var bar = $( "<div></div>" ).appendTo( "body" ).
2
3  // get the current value
4  alert( bar.progressBar( "value" ) );
5
6  // update the value
7  bar.progressBar( "value", 50 );
8
9  // get the current value again
10 alert( bar.progressBar( "value" ) );
```

### Note Executing methods by passing the method name to the same jQuery function that was used to initialize the plugin may seem odd. This is done to prevent pollution of the jQuery namespace while maintaining the ability to chain method calls.

## Working with Widget Options

One of the methods that is automatically available to our plugin is the option method. The option method allows you to get and set options after initialization. This method works exactly like jQuery's `css` and `attr` methods: you can pass just a name to use it as a setter, a name and value to use it as a single setter, or a hash of name/value pairs to set multiple values. When used as a getter, the plugin will return the current value of the option that corresponds to the name that was passed in. When used as a setter, the plugin's `_setOption` method will be called for each option that is being set. We can specify a `_setOption` method in our plugin to react to option changes.

Responding when an option is set

```
1  $.widget( "nmk.progressbar", {
2
3      options: {
4
5          value: 0
6
7      },
8
9      _create: function() {
10
11          this.element.addClass("progressbar");
12
13          this._update();
14
15      },
16
17      _setOption: function( key, value ) {
18
19          this.options[ key ] = value;
20
21          this._update();
22
```

```
23     },
24
25     _update: function() {
26
27         var progress = this.options.value + "%";
28
29         this.element.text( progress );
30
31     }
32
33 });
```

## Adding Callbacks

One of the easiest ways to make your plugin extensible is to add callbacks so users can react when the state of your plugin changes. We can see below how to add a callback to our progress bar to signify when the progress has reached 100%. The `_trigger` method takes three parameters: the name of the callback, a native event object that initiated the callback, and a hash of data relevant to the event. The callback name is the only required parameter, but the others can be very useful for users who want to implement custom functionality on top of your plugin. For example, if we were building a draggable plugin, we could pass the native `mousemove` event when triggering a drag callback; this would allow users to react to the drag based on the x/y coordinates provided by the event object.

Providing callbacks for user extension

```
1  $.widget( "nmk.progressbar", {
2
3      options: {
4
5          value: 0
6
7      },
```

```

8
9     _create: function() {
10
11         this.element.addClass("progressbar");
12
13         this._update();
14     },
15
16     _setOption: function( key, value ) {
17
18         this.options[ key ] = value;
19
20         this._update();
21     },
22
23     _update: function() {
24
25         var progress = this.options.value + "%";
26
27         this.element.text( progress );
28
29         if ( this.options.value == 100 ) {
30
31             this._trigger( "complete", null, { value:
32
33         }
34
35     }
36
37 });

```

Callback functions are essentially just additional options, so you can get and set them just like any other option. Whenever a callback is executed, a corresponding event is triggered as well. The event type is determined by concatenating the plugin name and the callback name. The callback and event both receive the

same two parameters: an event object and a hash of data relevant to the event, as we'll see below.

If your plugin has functionality that you want to allow the user to prevent, the best way to support this is by creating cancelable callbacks. Users can cancel a callback, or its associated event, the same way they cancel any native event: by calling `event.preventDefault()` or using `return false`. If the user cancels the callback, the `_trigger` method will return false so you can implement the appropriate functionality within your plugin.

Binding to widget events

```
1  var bar = $("<div></div>").appendTo("body").
2
3      complete: function( event, data ) {
4
5          alert("Callbacks are great!");
6
7      }
8
9  }).bind( "progressbarcomplete", function( ev
10
11      alert("Events bubble and support many hanc
12
13      alert( "The progress bar value is " + data
14
15  });
16
17  bar.progressbar( "option", "value", 100 );
```

## The Widget Factory: Under the Hood

When you call `jQuery.widget`, it creates a constructor function for your plugin and sets the object literal that you pass in as the prototype for your plugin instances. All of the functionality that automatically gets added to your plugin comes from a base widget

prototype, which is defined as `jQuery.Widget.prototype`. When a plugin instance is created, it is stored on the original DOM element using `jQuery.data`, with the plugin name as the key.

Because the plugin instance is directly linked to the DOM element, you can access the plugin instance directly instead of going through the exposed plugin method if you want. This will allow you to call methods directly on the plugin instance instead of passing method names as strings and will also give you direct access to the plugin's properties.

```
1  var bar = $( "<div></div>" )
2    .appendTo( "body" )
3    .progressbar()
4    .data( "progressbar" );
5
6  // call a method directly on the plugin inst
7  bar.option( "value", 50 );
8
9  // access properties on the plugin instance
10 alert( bar.options.value );
```

One of the biggest benefits of having a constructor and prototype for a plugin is the ease of extending the plugin. By adding or modifying methods on the plugin's prototype, we can modify the behavior of all instances of our plugin. For example, if we wanted to add a method to our progress bar to reset the progress to 0% we could add this method to the prototype and it would instantly be available to be called on any plugin instance.

```
1  $.nmk.progressbar.prototype.reset = function(
2
3    this._setOption( "value", 0 );
4
5  };
```

## Cleaning Up

In some cases, it will make sense to allow users to apply and then later unapply your plugin. You can accomplish this via the `destroy` method. Within the `destroy` method, you should undo anything your plugin may have done during initialization or later use. The `destroy` method is automatically called if the element that your plugin instance is tied to is removed from the DOM, so this can be used for garbage collection as well. The default `destroy` method removes the link between the DOM element and the plugin instance, so it's important to call the base function from your plugin's `destroy` method.

Adding a `destroy` method to a widget

```
1  $.widget( "nmk.progressbar", {
2
3    options: {
4      value: 0
5    },
6
7    _create: function() {
8
9      this.element.addClass("progressbar");
10
11     this._update();
12
13   },
14
15   _setOption: function( key, value ) {
16
17     this.options[ key ] = value;
18
19     this._update();
20
21   }
22 }
```

```

23     },
24
25     _update: function() {
26
27         var progress = this.options.value + "%";
28
29         this.element.text( progress );
30
31         if ( this.options.value === 100 ) {
32
33             this._trigger( "complete", null, { value:
34
35         }
36
37     },
38
39     destroy: function() {
40
41         this.element
42             .removeClass("progressbar")
43             .text("");
44
45         // call the base destroy function
46         $.Widget.prototype.destroy.call( this );
47
48     }
49
50 });

```

## Conclusion

The widget factory is only one way of creating stateful plugins. There are a few different models that can be used and each have their own advantages and disadvantages. The widget factory solves lots of common problems for you and can greatly improve productivity, it also greatly improves code reuse, making it a great fit

for jQuery UI as well as many other stateful plugins.

# Performance

- [Append Outside of Loops](#)
- [Cache Length During Loops](#)
- [Detach Elements to Work with Them](#)
- [Don't Act on Absent Elements](#)
- [Optimize Selectors](#)
- [Use Stylesheets for Changing CSS on Many Elements](#)
- [Variable Definition](#)
- [Don't Treat jQuery as a Black Box](#)

# Append Outside of Loops

Touching the DOM comes at a cost; if you're appending a lot of elements to the DOM, you will want to append them all at once, rather than one at a time. This is common problem when appending elements within a loop.

```
1 $.each( myArray, function( i, item ) {
2
3     var newListItem = "<li>" + item + "</li>";
4
5     $("#ballers").append( newListItem );
6
7 });
```

One common technique is to leverage a document fragment. During each iteration of the loop, you append the element to the fragment rather than the DOM element. After the loop, just append the fragment to the DOM element.

```
1 var frag = document.createDocumentFragment()
2
3 $.each( myArray, function( i, item ) {
4
5     var newListItem = document.createElement("li");
6     var itemText = document.createTextNode( item );
7
8     newListItem.appendChild( itemText );
9
10    frag.appendChild( newListItem );
11
12 });
13
14 $("#ballers")[ 0 ].appendChild( frag );
```

Another technique, which is quite simple, is to build up a string during each iteration of the loop. After the loop, just set the html of the DOM element to that string.

```
1  var myHtml = "";
2
3  $.each( myArray, function( i, item ) {
4
5      myHtml += "<li>" + item + "</li>";
6
7  });
8
9  $("#ballers").html( myHtml );
```

There are of course other techniques you could certainly test out; a great way to test the performance of these is through a site called [jsperf](#). This site allows you to benchmark each technique and visually see how it performs across all the browsers.

# Cache Length During Loops

In a for loop, don't access the length property of an array every time; cache it beforehand.

```
1  var myLength = myArray.length;
2
3  for ( var i = 0; i < myLength; i++ ) {
4
5      // do stuff
6
7  }
```

# Detach Elements to Work with Them

The DOM is slow; you want to avoid manipulating it as much as possible. jQuery introduced `$.fn.detach` in version 1.4 to help address this issue, allowing you to remove an element from the DOM while you work with it.

```
1  var $table = $("#myTable");
2  var $parent = $table.parent();
3
4  $table.detach();
5
6  // ... add lots and lots of rows to table
7  $parent.append( $table );
```

# Don't Act on Absent Elements

jQuery won't tell you if you're trying to run a whole lot of code on an empty selection — it will proceed as though nothing's wrong. It's up to you to verify that your selection contains some elements.

```
1 // BAD: this runs three functions
2 // before it realizes there's nothing
3 // in the selection
4 $("#nosuchthing").slideUp();
5
6 // Better
7 var $mySelection = $("#nosuchthing");
8
9 if ( $mySelection.length ) {
10
11     $mySelection.slideUp();
12
13 }
14
15 // BEST: add a doOnce plugin
16 jQuery.fn.doOnce = function( func ){
17
18     this.length && func.apply( this );
19
20     return this;
21
22 }
23
24 $("li.cartitems").doOnce(function() {
25
26     // make it ajax! \o/
27
28 });
```

This guidance is especially applicable for jQuery UI widgets, which have a lot of overhead even when the selection doesn't contain elements.

# Optimize Selectors

Selector optimization is less important than it used to be, as more browsers implement `document.querySelectorAll()` and the burden of selection shifts from jQuery to the browser. However, there are still some tips to keep in mind.

## ID-Based Selectors

Beginning your selector with an ID is always best.

```
1 // fast
2 $("#container div.robotarm");
3
4 // super-fast
5 $("#container").find("div.robotarm");
```

The `$.fn.find` approach is faster because the first selection is handled without going through the Sizzle selector engine — ID-only selections are handled using `document.getElementById()`, which is extremely fast because it is native to the browser.

## Specificity

Be specific on the right-hand side of your selector, and less specific on the left.

```
1 // unoptimized
2 $("div.data .gonzalez");
3
4 // optimized
5 $(".data td.gonzalez");
```

Use `tag.class` if possible on your right-most selector, and just tag or just `.class` on the left.

## Avoid excessive specificity.

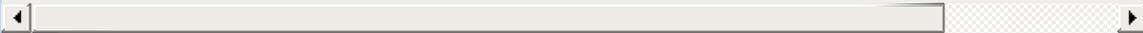
```
1 | $(".data table.attendees td.gonzalez");  
2 |  
3 | // better: drop the middle if possible  
4 | $(".data td.gonzalez");
```

A "flatter" DOM also helps improve selector performance, as the selector engine has fewer layers to traverse when looking for an element.

## Avoid the Universal Selector

Selections that specify or imply that a match could be found anywhere can be very slow.

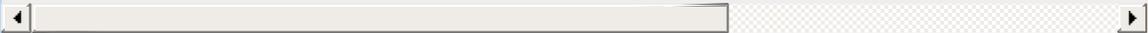
```
1  $(".buttons > *"); // extremely expensive
2  $(".buttons").children(); // much better
3
4  $(".gender :radio"); // implied universal se
5  $(".gender *:radio"); // same thing, explicit
6  $(".gender input:radio"); // much better
```



# Use Stylesheets for Changing CSS on Many Elements

If you're changing the CSS of more than 20 elements using `$.fn.css`, consider adding a style tag to the page instead for a nearly 60% increase in speed.

```
1 // fine for up to 20 elements, slow after tha
2 $("a.swedberg").css("color", "#asd123");
3
4 $("<style type=\"text/css\">a.swedberg { colc
5     .appendTo("head");
```



# Variable Definition

Variables can be defined in one statement instead of several.

```
1 // old & busted
2 var test = 1;
3 var test2 = function() { ... };
4 var test3 = test2( test );
5
6 // new hotness
7 var test = 1,
8     test2 = function() { ... },
9     test3 = test2( test );
```

In self-executing functions, variable definition can be skipped all together.

```
1 (function( foo, bar ) {
2     // ...
3 })( 1, 2 );
```

# Don't Treat jQuery as a Black Box

Use the source as your documentation. Bookmark <http://bit.ly/jqsource> and refer to it often.

# Code Organization

Understanding the basic mechanics is one thing, but the essence of building applications is understanding how to organize code so that it is navigable and well-encapsulated instead of a whole slew of global functions.

- [Code Organization Concepts](#)
- [Beware Anonymous Functions](#)
- [Keep Things DRY](#)
- [Feature & Browser Detection](#)
- [Deferreds](#)
  - [jQuery Deferreds](#)
  - [Deferred examples](#)

# Code Organization Concepts

When you move beyond adding simple enhancements to your website with jQuery and start developing full-blown client-side applications, you need to consider how to organize your code. In this chapter, we'll take a look at various code organization patterns you can use in your jQuery application and explore the RequireJS dependency management and build system.

## Key Concepts

Before we jump into code organization patterns, it's important to understand some concepts that are common to all good code organization patterns.

- Your code should be divided into units of functionality — modules, services, etc. Avoid the temptation to have all of your code in one huge `$( document ).ready()` block. This concept, loosely, is known as encapsulation.
- Don't repeat yourself. Identify similarities among pieces of functionality, and use inheritance techniques to avoid repetitive code.
- Despite jQuery's DOM-centric nature, JavaScript applications are not all about the DOM. Remember that not all pieces of functionality need to — or should — have a DOM representation.
- Units of functionality should be [loosely coupled](#), that is, a unit of functionality should be able to exist on its own, and communication between units should be handled via a messaging system such as custom events or pub/sub. Stay away from direct communication between units of functionality whenever possible.

The concept of loose coupling can be especially troublesome to developers making their first foray into complex applications, so be mindful of this as you're getting started.

## Encapsulation

The first step to code organization is separating pieces of your application into distinct pieces; sometimes, even just this effort is sufficient to lend

## The Object Literal

An object literal is perhaps the simplest way to encapsulate related code. It doesn't offer any privacy for properties or methods, but it's useful for eliminating anonymous functions from your code, centralizing configuration options, and easing the path to reuse and refactoring.

```
1 // An object literal
2 var myFeature = {
3   myProperty: "hello",
4   myMethod: function() {
5     console.log( myFeature.myProperty );
6   },
7   init: function( settings ) {
8     myFeature.settings = settings;
9   },
10  readSettings : function() {
11    console.log( myFeature.settings );
12  }
13 };
14
15 myFeature.myProperty === "hello"; // true
16
17 myFeature.myMethod(); // "hello"
18
19 myFeature.init({
20   foo: "bar"
21 });
22
23 myFeature.readSettings(); // { foo: "bar" }
```

The object literal above is simply an object assigned to a variable. The object has one property and several methods. All of the properties and methods are public, so any part of your application can see the properties and call methods on the object. While there is an `init` method, there's nothing requiring that it be called before the object is functional.

How would we apply this pattern to jQuery code? Let's say that we had this code written in the traditional jQuery style:

```
1 // clicking on a list item loads some content
2 // using the list item's ID and hides content
3 // in sibling list items
4 $( document ).ready(function() {
5
6     $("#myFeature li").append("<div/>").click(
7
8         var $this = $( this );
9
10        var $div = $this.find("div");
11
12        $div.load( "foo.php?item=" + $this.attr(
13
14            $div.show();
15
16            $this.siblings().find("div").hide();
17
18        }
19
20    );
21
22 });
23
24 });
```

If this were the extent of our application, leaving it as-is would be fine. On the other hand, if this was a piece of a larger application, we'd do well to keep this functionality separate from unrelated functionality. We might also want to move the URL out of the code and into a configuration area. Finally, we might want to break up the chain to make it easier to modify pieces of the functionality later.

```
1 // Using an object literal for a jQuery feat
2 var myFeature = {
3
4   init : function( settings ) {
5
6     myFeature.config = {
7       $items : $("#myFeature li"),
8       $container : $("<div class='container'
9         urlBase : "/foo.php?item="
10    };
11
12    // allow overriding the default config
13    $.extend( myFeature.config, settings );
14
15    myFeature.setup();
16
17  },
18
19  setup : function() {
20
21    myFeature.config.$items.each( myFeature.
22
23  },
24
25  createContainer : function() {
26
27    var $i = $( this );
28
29    var $c = myFeature.config.$container.clc
30
```

```
31     $i.data( "container", $c );
32
33     },
34
35     buildUrl : function() {
36
37         return myFeature.config.urlBase + myFeature.$currentItem.url;
38     },
39
40
41     showItem : function() {
42
43         var myFeature.$currentItem = $( this );
44
45         myFeature.getContent( myFeature.showContent );
46
47     },
48
49     getContent : function( callback ) {
50
51         var url = myFeature.buildUrl();
52
53         myFeature.$currentItem.data("container").load( url );
54
55     },
56
57     showContent : function() {
58
59         myFeature.$currentItem.data("container").show();
60
61         myFeature.hideContent();
62
63     },
64
65     hideContent : function() {
66
67         myFeature.$currentItem.siblings().each(function
```

```
68
69         $( this ).data("container").hide();
70
71     });
72
73 }
74
75 };
76
77 $( document ).ready( myFeature.init );
```

The first thing you'll notice is that this approach is obviously far longer than the original -- again, if this were the extent of our application, using an object literal would likely be overkill. Assuming it's not the extent of our application, though, we've gained several things:

- We've broken our feature up into tiny methods. In the future, if we want to change how content is shown, it's clear where to change it. In the original code, this step is much harder to locate.
- We've eliminated the use of anonymous functions.
- We've moved configuration options out of the body of the code and put them in a central location.
- We've eliminated the constraints of the chain, making the code easier to refactor, remix, and rearrange.

For non-trivial features, object literals are a clear improvement over a long stretch of code stuffed in a `$( document ).ready()` block, as they get us thinking about the pieces of our functionality. However, they aren't a whole lot more advanced than simply having a bunch of function declarations inside of that `$( document ).ready()` block.

## The Module Pattern

The module pattern overcomes some of the limitations of the object literal, offering privacy for variables and functions while exposing a

public API if desired.

```
1 // The module pattern
2 var feature = (function() {
3
4     // private variables and functions
5     var privateThing = "secret";
6     var publicThing = "not secret";
7     var changePrivateThing = function() {
8         privateThing = "super secret";
9     };
10
11     var sayPrivateThing = function() {
12         console.log( privateThing );
13         changePrivateThing();
14     };
15
16     // public API
17     return {
18         publicThing: publicThing,
19         sayPrivateThing: sayPrivateThing
20     };
21
22 })();
23
24 feature.publicThing; // "not secret"
25
26
27 // logs "secret" and changes the value
28 // of privateThing
29 feature.sayPrivateThing();
```

In the example above, we self-execute an anonymous function that returns an object. Inside of the function, we define some variables. Because the variables are defined inside of the function, we don't have access to them outside of the function unless we put them in

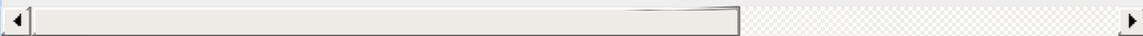
the return object. This means that no code outside of the function has access to the `privateThing` variable or to the `changePrivateThing` function. However, `sayPrivateThing` does have access to `privateThing` and `changePrivateThing`, because both were defined in the same scope as `sayPrivateThing`.

This pattern is powerful because, as you can gather from the variable names, it can give you private variables and functions while exposing a limited API consisting of the returned object's properties and methods.

Below is a revised version of the previous example, showing how we could create the same feature using the module pattern while only exposing one public method of the module, `showItemByIndex()`.

```
1 // Using the module pattern for a jQuery fea
2 $( document ).ready(function() {
3
4     var feature = (function() {
5
6         var $items = $("#myFeature li");
7         var $container = $("<div class='containe
8         var $currentItem = null;
9         var urlBase = "/foo.php?item=";
10        var createContainer = function() {
11            var $i = $( this );
12            var $c = $container.clone().appendTo(
13                $i.data( "container", $c );
14        },
15        buildUrl = function() {
16            return urlBase + $currentItem.attr("ic
17        },
18        showItem = function() {
19            $currentItem = $( this );
20            getContent( showContent );
21        },
22        showItemByIndex = function( idx ) {
23            $.proxy( showItem, $items.get( idx ) )
```

```
24     },
25     getContent = function( callback ) {
26         $currentItem.data("container").load( buildUrl(),
27     },
28     showContent = function() {
29         $currentItem.data("container").show();
30         hideContent();
31     },
32     hideContent = function() {
33         $currentItem.siblings().each(function() {
34             $( this ).data("container").hide();
35         });
36     };
37     $items.each( createContainer ).click( showItem );
38
39     return {
40         showItemByIndex: showItemByIndex
41     };
42
43     })();
44
45     feature.showItemByIndex( 0 );
46 });
```



# Beware Anonymous Functions

Anonymous functions bound everywhere are a pain. They're difficult to debug, maintain, test, or reuse. Instead, use an object literal to organize and name your handlers and callbacks.

```
1 // BAD
2 $( document ).ready(function() {
3
4     $("#magic").click(function( event ) {
5
6         $("#yayeffects").slideUp(function() {
7
8             // ...
9
10        });
11
12    });
13
14    $("#happiness").load( url + " #unicorns",
15
16        // ...
17
18    });
19
20 });
21
22 // BETTER
23 var PI = {
24
25     onReady : function() {
26
27         $("#magic").click( PI.candyMtn );
28
29         $("#happiness").load( PI.url + " #unicor
```

```
30
31     },
32
33     candyMtn : function( event ) {
34
35         $("#yayeffects").slideUp( PI.slideCb );
36
37     },
38
39     slideCb : function() { ... },
40
41     unicornCb : function() { ... }
42
43 };
44
45 $( document ).ready( PI.onReady );
```

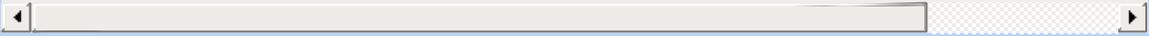


# Keep Things DRY

Don't repeat yourself; if you're repeating yourself, you're doing it wrong.

```
1 // BAD
2 if ( $eventfade.data("currently") !== "showi
3
4     $eventfade.stop();
5
6 }
7
8 if ( $eventhover.data("currently") !== "show
9
10    $eventhover.stop();
11
12 }
13
14 if ( $spans.data("currently") !== "showing"
15
16    $spans.stop();
17
18 }
19
20 // GOOD!!
21 var $elems = [ $eventfade, $eventhover, $spa
22
23 $.each( $elems, function( i, elem ) {
24
25     if ( elem.data("currently") !== "showing"
26
27         elem.stop();
28
29 }
30
```

```
31 | });
```



# Feature & Browser Detection

## Can I Use This Browser Feature?

There are a couple of common ways to check whether or not a particular feature is supported by a user's browser:

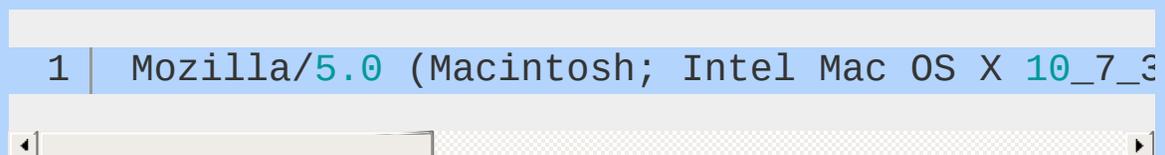
- Browser Detection
- Specific Feature Detection

In general, we recommend specific feature detection. Let's look at why.

## Browser Detection

Browser detection is a method where the browser's User Agent (UA) string is checked for a particular pattern unique to a browser family or version. For example, this is Chrome 18's UA string on Mac OS X Lion:

```
1 | Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_3
```



Browser UA detection may check this string for something like "Chrome" or "Chrome/18" or any other part the developer feels identifies the browser they intend to target.

While this seems to be an easy solution, there are several problems:

**Other browsers other than your target may have the same issue.**

If we target a specific browser for different functionality, we implicitly exclude any browser we did not account for. This is also not future-proof. If the browser we target receives a bug fix or change, we may not be able to discern between a 'working' and 'non-working'

UA string. We may also need to update our test for each new release. This isn't a maintainable solution.

### **User Agents are unreliable.**

User Agents are set by the client browser. In the early days of the web, browsers would mimic each others' UA strings in order to bypass exactly this type of detection. It is still possible that a browser with very different capabilities may mimic just the portion of the UA string you're targeting.

The UA string is also user-configurable. While the user may change this string, the browser's feature support remains the same.

In general, we do not recommend UA string-based feature detection.

## **Specific Feature Detection**

Specific feature detection checks if a specific feature is available, instead of developing against a specific browser. This way, developers can write their code for two cases: the browser **does** support said feature, or the browser **does not** support said feature.

Developing against specific features, instead of developing against a specific browser, not only clears up the peripheral logic of your application, but also makes your job easier as a developer.

We recommend specific feature detection over UA string detection.

Now how would you go about doing that?

## **How to go about feature detection**

There are several ways to go about feature detection:

- Straight JavaScript
- \$.support
- A Helper Library

### **Straight JavaScript**

Let's take a look at how to check whether or not a `<canvas>` element exists in a specific browser, without using a helper library. We do this by specifically querying whether the method or property exists:

```
1 // We want to show a graph in browsers that
2 // but a data table in browsers that don't.
3 var elem = document.createElement("canvas");
4
5 if ( elem.getContext && elem.getContext("2d"
6
7     showGraph();
8
9 } else {
10
11     showTable();
12
13 }
```

This is a very simple way to provide conditional experiences, depending on the features present in the user's browser. We can extract this into a helper function for reuse, but still have to write a test for every feature we're concerned about. This can be time-consuming and error-prone.

What if someone else wrote all of that for us?

### **\$.support**

jQuery performs many tests to determine feature support to allow cross-browser use of many of the features we've come to love. jQuery's internal feature detection can be accessed through [jQuery.support](#).

However, we do not recommend this for general use. As the API page says:

*Since jQuery requires these tests internally,*

*they must be performed on every page load. Although some of these properties are documented below, they are not subject to a long deprecation/removal cycle and may be removed once internal jQuery code no longer needs them.*

This detection may be removed from jQuery without notice. That's reason enough not to use it. What other options do we have?

## A Helper Library

Thankfully, there are some great helper libraries (like [Modernizr](#)) that provide a simple, high-level API for determining if a browser has a specific feature available or not.

For example, utilizing Modernizr, we are able to do the same canvas detection test with this code:

```
1  if ( Modernizr.canvas ) {  
2  
3      showGraphWithCanvas();  
4  
5  } else {  
6  
7      showTable();  
8  
9  }
```

That's it. Easy.

## Performance Considerations

So, while the Modernizr syntax is great, it can end up being quite cumbersome to have several conditionals. Secondly, we're sending

the code for both conditions to every browser, regardless if we'll need it or not.

The Modernizr object exposes a `load()` method that many prefer over the syntax mentioned previously. This is due to the another library that Modernizr now uses internally: [yepnope](#). Testing for canvas can now become something like this:

```
1 Modernizr.load({
2   test: Modernizr.canvas,
3   yep : "canvas.js",
4   nope: "canvas-polyfill.js"
5 });
```

Using the `load` method allows us to send only the required polyfills and code over the wire. You can also pass an array of objects as an argument to `.load()`, and it will serve the correct files to the correct audience.

Additionally, Modernizr has a [production build configurator](#) that allows you to specify exactly what parts of Modernizr you want to include and exclude the parts you don't.

## Other Resources

### Feature Detection Tools

- [modernizr](#) - conditionally check to see if a specific feature is available in a browser
- [html5please](#) - use the new and shiny responsibly
- [html5please api](#) - an API you can query to see how good (or bad) support for a specific feature is.
- [caniuse](#) - browser compatibility tables for HTML5, CSS3, SVG, etc...
- [yepnope](#) - conditional polyfill loader

### Helpful Articles

- [Browser Feature Detection](#)
- [Using Modernizr to detect HTML5 and CSS3 browser support](#)
- [polyfilling the html5 gap](#) by Addy Osmani
- [Feature, Browser, and Form Factor Detection: It's Good for the Environment](#) by Michael Mahemoff

# Deferreds

At a high-level, deferreds can be thought of as a way to represent asynchronous operations which can take a long time to complete. They're the asynchronous alternative to blocking functions and the general idea is that rather than your application blocking while it awaits some request to complete before returning a result, a deferred object can instead be returned immediately. You can then attach callbacks to the deferred object: they will be called once the request has actually completed.

## Promises

In its most basic form, a 'promise' is a model that provides a solution for the concept of deferred (or future) results in software engineering. The main idea behind it is something we've already covered: rather than executing a call which may result in blocking, we instead return a promise for a future value that will eventually be satisfied.

If it helps to have an example here, consider that you are building a web application which heavily relies on data from a third party API. A common problem that's faced is having an unknown knowledge of the API server's latency at a given time so it's possible that other parts of your application may be blocked from running until a result from it is returned. Deferreds provide a better solution to this problem, one which is void of 'blocking' effects and completely decoupled.

The [Promise/A](#) proposal defines a method called 'then' that can be used to register callbacks to a promise and, thus, get the future result when it is available. The pseudo-code for dealing with a third party API that returns a promise may look like:

```
1  promise = callToAPI( arg1, arg2, ... );
2
3  promise.then(function( futureValue ) {
4      /* handle futureValue */
5  });
6
7  promise.then(function( futureValue ) {
8      /* do something else */
9  });
```

Furthermore, a promise can actually end up being in two different states:

- resolved: in which case data is available
- rejected: in which case something went wrong and no value is

available

Thankfully, the 'then' method accepts two parameters: one for when the promise was resolved, another for when the promise was rejected. If we get back to pseudo-code, we may do things like:

```
1 | promise.then( function( futureValue ) {  
2 |     /* we got a value */  
3 | } , function() {  
4 |     /* something went wrong */  
5 | } );
```

In the case of certain applications, it is necessary to have several results returned before your application can continue at all (for example, displaying a dynamic set of options on a screen before a user is able to select the option that interests them). Where this is the case, a method called 'when' exists, which can be used to perform some action once all the promises have been fully fulfilled:

```
1 | when(  
2 |     promise1,  
3 |     promise2,  
4 |     ...  
5 | ).then(function( futureValue1, futureValue2,  
6 |     /* all promises have completed and are re  
7 | });
```

A good example is a scenario where you may have multiple concurrent animations that are being run. Without keeping track of each callback firing on completion, it can be difficult to truly establish once all your animations have finished running. Using promises and 'when' however this is very straightforward as each of your animations can effectively say 'we promise to let you know once we're done'. The compounded result of this means it's a trivial process to execute a single callback once the animations are done.

For example:

```
1  var promise1 = $("#id1").animate().promise();
2  var promise2 = $("#id2").animate().promise();
3  when(
4      promise1,
5      promise2
6  ).then(function(){
7      /* once both animations have completed
8         we can then run our additional logic
9  });
```

This means that one can basically write non-blocking logic that can be executed without synchronization. Rather than directly passing callbacks to functions, something which can lead to tightly coupled interfaces, using promises allows one to separate concerns for code that is synchronous or asynchronous.

- [jQuery Deferreds](#)
- [Deferred examples](#)

# jQuery Deferreds

## jQuery Deferreds

Deferreds were added as a part of a large rewrite of the ajax module, led by Julian Auborg following the CommonJS Promises/A design. Whilst 1.5 and above include deferred capabilities, former versions of jQuery had `jQuery.ajax()` accept callbacks that would be invoked upon completion or error of the request, but suffered from heavy coupling - the same principle that would drive developers using other languages or toolkits to opt for deferred execution.

In practice what jQuery's version provides you with are several enhancements to the way callbacks are managed, giving you significantly more flexible ways to provide callbacks that can be invoked whether the original callback dispatch has already fired or not. It is also worth noting that jQuery's Deferred object supports having multiple callbacks bound to the outcome of particular tasks (and not just one) where the task itself can either be synchronous or asynchronous.

At the heart of jQuery's implementation is `jQuery.Deferred` - a chainable constructor which is able to create new deferred objects that can check for the existence of a promise to establish whether the object can be observed. It can also invoke callback queues and pass on the success of synchronous and asynchronous functions. It's quite essential to note that the default state of any Deferred object is unresolved. Callbacks which may be added to it through `.then()` or `.fail()` are queued up and get executed later on in the process.

You are able to use Deferred objects in conjunction with the promise concept of `when()`, implemented in jQuery as `$.when()` to wait for all of the Deferred object's requests to complete executing (ie. for all of the promises to be fulfilled). In technical terms, `$.when()` is effectively a way to execute callbacks based on any number of promises that represent asynchronous events.

An example of `$.when()` accepting multiple arguments can be seen below in conjunction with `.then()`:

```
1 | function successFunc() {
```

```

2     console.log( "success!" );
3 }
4
5 function failureFunc(){
6     console.log( "failure!" );
7 }
8
9 $.when(
10    $.ajax( "/main.php" ),
11    $.ajax( "/modules.php" ),
12    $.ajax( "/lists.php" )
13 ).then( successFunc, failureFunc );

```

The \$.when() implementation offered in jQuery is quite interesting as it not only interprets deferred objects, but when passed arguments that are not deferreds, it treats these as if they were resolved deferreds and executes any callbacks (doneCallbacks) right away. It is also worth noting that jQuery's deferred implementation, in addition to exposing deferred.then(), a jQuery promise also supports the deferred.done() and deferred.fail() methods which can also be used to add callbacks to the deferred's queues.

We will now take a look at a code example that utilizes many of the deferred features mentioned in the table presented earlier. Here is a very basic application that consumes (1) an external news feed and (2) a reactions feed for pulling in the latest comments via \$.get() (which will return a promise). The application also has a function (prepareInterface()) which returns a promise to complete animating our containers for both the news and reactions.

```

1 function getLatestNews() {
2     return $.get( "latestNews.php", function
3         console.log( "news data received" );
4         $( ".news" ).html(data);
5     } );
6 }

```

```
7
8  function getLatestReactions() {
9      return $.get( "latestReactions.php", function
10         console.log( "reactions data received"
11         $( ".reactions" ).html(data);
12     } );
13 }
14
15 function prepareInterface() {
16     return $.Deferred(function( dfd ) {
17         var latest = $( ".news, .reactions" );
18         latest.slideDown( 500, dfd.resolve );
19         latest.addClass( "active" );
20     }).promise();
21 }
22
23 $.when(
24     getLatestNews(),
25     getLatestReactions(),
26     prepareInterface()
27 ).then(function(){
28     console.log( "fire after requests succeed"
29 }).fail(function(){
30     console.log( "something went wrong!" );
31 });
```



# Deferred examples

## Further Deferreds examples

Deferreds are used behind the hood in Ajax but it doesn't mean they can't also be used elsewhere. This section describes situations where deferreds will help abstract away asynchronous behaviour and decouple our code.

## Caching

### Asynchronous cache

When it comes to asynchronous tasks, caching can be a bit demanding since you have to make sure a task is only performed once for a given key. As a consequence, the code has to somehow keep track of inbound tasks.

```
1 | $.cachedGetScript( url, callback1 );  
2 | $.cachedGetScript( url, callback2 );
```

The caching mechanism has to make sure the url is only requested once even if the script isn't in cache yet. This shows some logic to keep track of callbacks bound to a given url in order for the cache system to properly handle both complete and inbound requests.

```
1 | var cachedScriptPromises = {};  
2 |  
3 | $.cachedGetScript = function( url, callback  
4 |     if ( !cachedScriptPromises[ url ] ) {  
5 |         cachedScriptPromises[ url ] = $.Defer  
6 |             $.getScript( url ).then( defer.r  
7 |             }).promise();  
8 |     }  
9 |     return cachedScriptPromises[ url ].done(  
10 | );
```

One promise is cached per url. If there is no promise for the given url yet, then a deferred is created and the request is issued. If it already exists, however, the callback is attached to the existing deferred. The big advantage of this solution is that it will handle both complete and inbound requests transparently. Another advantage is that a deferred-based cache will deal with failure gracefully. The promise will end up rejected which can be tested for by providing an error callback:

```
1 | $.cachedGetScript( url ).then( successCallback
```

### Generic asynchronous cache

It is also possible to make the code completely generic and build a cache factory that will abstract out the actual task to be performed when a key isn't in the cache yet:

```
1 | $.createCache = function( requestFunction )
2 |     var cache = {};
3 |     return function( key, callback ) {
4 |         if ( !cache[ key ] ) {
5 |             cache[ key ] = $.Deferred(function()
6 |                 requestFunction( defer, key
7 |                 }).promise();
8 |         }
9 |         return cache[ key ].done( callback )
10 |     };
11 | }
```

Now that the request logic is abstracted away, cachedGetScript can be rewritten as follows:

```
1 | $.cachedGetScript = $.createCache(function( c
2 |     $.getScript( url ).then( defer.resolve, c
```

```
3 | });
```

This will work because every call to `createCache` will create a new cache repository and return a new cache-retrieval function.

## Image loading

A cache can be used to ensure that the same image is not loaded multiple times.

```
1 | $.loadImage = $.createCache(function( defer,
2 |     var image = new Image();
3 |     function cleanup() {
4 |         image.onload = image.onerror = null;
5 |     }
6 |     defer.then( cleanup, cleanup );
7 |     image.onload = function() {
8 |         defer.resolve( url );
9 |     };
10 |     image.onerror = defer.reject;
11 |     image.src = url;
12 | });
```

Again, the following snippet:

```
1 | $.loadImage( "my-image.png" ).done( callback1
2 | $.loadImage( "my-image.png" ).done( callback2
```

will work regardless of whether `my-image.png` has already been loaded or not, or if it is actually in the process of being loaded.

## Caching Data API responses

API requests that are considered immutable during the lifetime of

your page are also perfect candidates. For instance, the following:

```
1 | $.searchTwitter = $.createCache(function( de
2 |     $.ajax({
3 |         url: "http://search.twitter.com/sear
4 |         data: {
5 |             q: query
6 |         },
7 |         dataType: "jsonp",
8 |         success: defer.resolve,
9 |         error: defer.reject
10 |     });
11 | });
```

will allow you to perform searches on Twitter and cache them at the same time:

```
1 | $.searchTwitter( "jQuery Deferred", callback1
2 | $.searchTwitter( "jQuery Deferred", callback2
```

## Timing

This deferred-based cache is not limited to network requests; it can also be used for timing purposes.

For instance, you may need to perform an action on the page after a given amount of time so as to attract the user's attention to a specific feature they may not be aware of or deal with a timeout (for a quiz question for instance). While `setTimeout` is good for most use-cases it doesn't handle the situation when the timer is asked for later, even after it has theoretically expired. We can handle that with the following caching system:

```
1 | var readyTime;
```

```

2
3  $(function() {
4      readyTime = jQuery.now();
5  });
6
7  $.afterDOMReady = $.createCache(function( defer, delay
8      delay = delay || 0;
9      $(function() {
10         var delta = $.now() - readyTime;
11         if ( delta >= delay ) {
12             defer.resolve();
13         } else {
14             setTimeout( defer.resolve, delay - delta )
15         }
16     });
17 });

```

The new `afterDOMReady` helper method provides proper timing after the DOM is ready while ensuring the bare minimum of timers will be used. If the delay is already expired, any callback will be called right away.

## One-time event

While jQuery offers all the event binding one may need, it can become a bit cumbersome to handle events that are only supposed to be dealt with once.

For instance, you may wish to have a button that will open a panel the first time it is clicked and leave it open afterwards or take special initialization actions the first time said button is clicked. When dealing with such a situation, one usually end up with code like this:

```

1  var buttonClicked = false;
2
3  $( "#myButton" ).click(function() {

```

```

4     if ( !buttonClicked ) {
5         buttonClicked = true;
6         initializeData();
7         showPanel();
8     }
9 });

```

then, later on, you may wish to take actions, but only if the panel is opened:

```

1     if ( buttonClicked ) {
2         /* perform specific action */
3     }

```

This is a very coupled solution. If you want to add some other action, you have to edit the bind code or just duplicate it all. If you don't, your only option is to test for buttonClicked and you may lose that new action because the buttonClicked variable may be false and your new code may never be executed.

We can do much better using deferreds (for simplification sake, the following code will only work for a single element and a single event type, but it can be easily generalized for full-fledged collections with multiple event types):

```

1     $.fn.bindOnce = function( event, callback )
2         var element = $( this[ 0 ] ),
3             defer = element.data( "bind_once_defer" );
4     if ( !defer ) {
5         defer = $.Deferred();
6         function deferCallback() {
7             element.unbind( event, deferCallback );
8             defer.resolveWith( this, arguments );
9         }
10    element.bind( event, deferCallback );
11    element.data( "bind_once_defer_" + event, defer );

```

```
12     }
13     return defer.done( callback ).promise();
14 };
```

The code works as follows:

- check if the element already has a deferred attached for the given event
- if not, create it and make it so it is resolved when the event is fired the first time around
- then attach the given callback to the deferred and return the promise

While the code is definitely more verbose, it makes dealing with the problem at hand much simpler in a compartmentalized and decoupled way. But let's define a helper method first:

```
1 $.fn.firstClick = function( callback ) {
2     return this.bindOnce( "click", callback );
3 };
```

Then the logic can be re-factored as follows:

```
1 var openPanel = $( "#myButton" ).firstClick()
2
3 openPanel.done( initializeData );
4 openPanel.done( showPanel );
```

If an action should be performed only when a panel is opened later on:

```
1 openPanel.done(function() {
```

```
2 |      /* perform specific action */
3 |    });
```

Nothing is lost if the panel isn't opened yet, the action will just get deferred until the button is clicked.

## Combining helpers

All of the samples above can seem a bit limited when looked at separately. However, the true power of promises comes into play when you mix them together.

### Requesting panel content on first click and opening said panel

Following is the code for a button that, when clicked, opens a panel. It requests its content over the wire and then fades the content in. Using the helpers defined earlier, it could be defined as:

```
1 | $( "#myButton" ).firstClick(function() {
2 |     var panel = $( "#myPanel" );
3 |     $.when(
4 |         $.get( "panel.html" ),
5 |         panel.slideDownPromise()
6 |     ).done(function( ajaxResponse ) {
7 |         panel.html( ajaxResponse[ 0 ] ).fadeI
8 |     });
9 | });
```

### Loading images in a panel on first click and opening said panel

Another possible goal is to have the panel fade in, only after the button has been clicked and after all of the images have been loaded.

The html code for this would look something like:

```
1 <div id="myPanel">
2   
3   
4   
5   
6 </div>
```

We use the data-src attribute to keep track of the real image location. The code to handle our use case using our promise helpers is as follows:

```
1 $( "#myButton" ).firstClick(function() {
2
3   var panel = $( "#myPanel" ),
4       promises = [];
5
6   $( "img", panel ).each(function() {
7     var image = $( this ),
8         src = element.attr( "data-src" );
9     if ( src ) {
10      promises.push(
11        $.loadImage( src ).then( func
12          image.attr( "src", src );
13        }, function() {
14          image.attr( "src", "error
15        } )
16      );
17    }
18  });
19
20  promises.push(
21    panel.slideDownPromise()
22  );
23
24  $.when.apply( null, promises ).done(function() {
25    panel.fadeIn();
```

```
26     });  
27   });
```

The trick here is to keep track of all the loadImage promises. We later join them with the panel slideDown animation using \$.when. So when the button is first clicked, the panel will slideDown and the images will start loading. Once the panel has finished sliding down and all the images have been loaded, then, and only then, will the panel fade in.

### Loading images on the page after a specific delay

In order to implement deferred image display on the entire page, the following format in HTML can be used.

```
1       
2       
3       
4     
```

What it says is pretty straight-forward:

- load image1.png and show it immediately for the third image and after one second for the first one
- load image2.png and show it after one second for the second image and after two seconds for the fourth image

```
1     $( "img" ).each(function() {  
2         var element = $( this ),  
3             src = element.attr( "data-src" ),  
4             after = element.attr( "data-after" )  
5         if ( src ) {  
6             $.when(  
7                 $.loadImage( src ),  
8                 $.afterDOMReady( after )
```

```

9         ).then(function() {
10             element.attr( "src", src );
11         }, function() {
12             element.attr( "src", "error.png" );
13         } ).done(function() {
14             element.fadeIn();
15         });
16     }
17 });

```

In order to delay the loading of the images themselves:

```

1     $( "img" ).each(function() {
2         var element = $( this ),
3             src = element.attr( "data-src" ),
4             after = element.attr( "data-after" )
5         if ( src ) {
6             $.afterDOMReady( after, function() {
7                 $.loadImage( src ).then(function
8                     element.attr( "src", src );
9                 }, function() {
10                    element.attr( "src", "error.
11                } ).done(function() {
12                    element.fadeIn();
13                });
14            } );
15        }
16    });

```

Here, after the delay to be fulfilled then the image is loaded. It can make a lot of sense when you want to limit the number or network requests on page load.

# jQuery UI

[jQuery UI](#) is a curated set of user interface interactions, effects, widgets, and themes built on top of the jQuery JavaScript Library. Whether you're building highly interactive web applications or you just need to add a date picker to a form control, jQuery UI is the perfect choice.

jQuery UI contains many widgets that maintain state and therefore have a slightly different usage pattern than typical jQuery plugins. All of jQuery UI's widgets use the same patterns, so if you learn how to use one, then you'll know how to use all of them.

- [Getting Started with jQuery UI](#)
- [How jQuery UI Works](#)
- [Theming jQuery UI](#)
  - [Using jQuery UI ThemeRoller](#)
  - [jQuery UI CSS Framework API](#)
  - [How To Write a Theme](#)
- [Widget Factory](#)
  - [Why Use the Widget Factory](#)
  - [How To Use the Widget Factory](#)

# Getting Started with jQuery UI

## What is jQuery UI?

jQuery UI is a widget and interaction library built on top of the jQuery JavaScript Library that you can use to build highly interactive web applications. This guide is designed to get you up to speed on how jQuery UI works. Follow along below to get started.

## Start by checking out the demos

To get a feel for what jQuery UI is capable of, check out the [UI Demos](#).

In the demos section, the navigation lists all of the interactions and widgets that jQuery UI offers. Choose an interaction or widget and you'll be presented with several demo configurations for that particular plugin. Each demo allows you to view source code, change themes, and the URL can always be bookmarked. For example, check out the [accordion widget's fill space demo page](#).

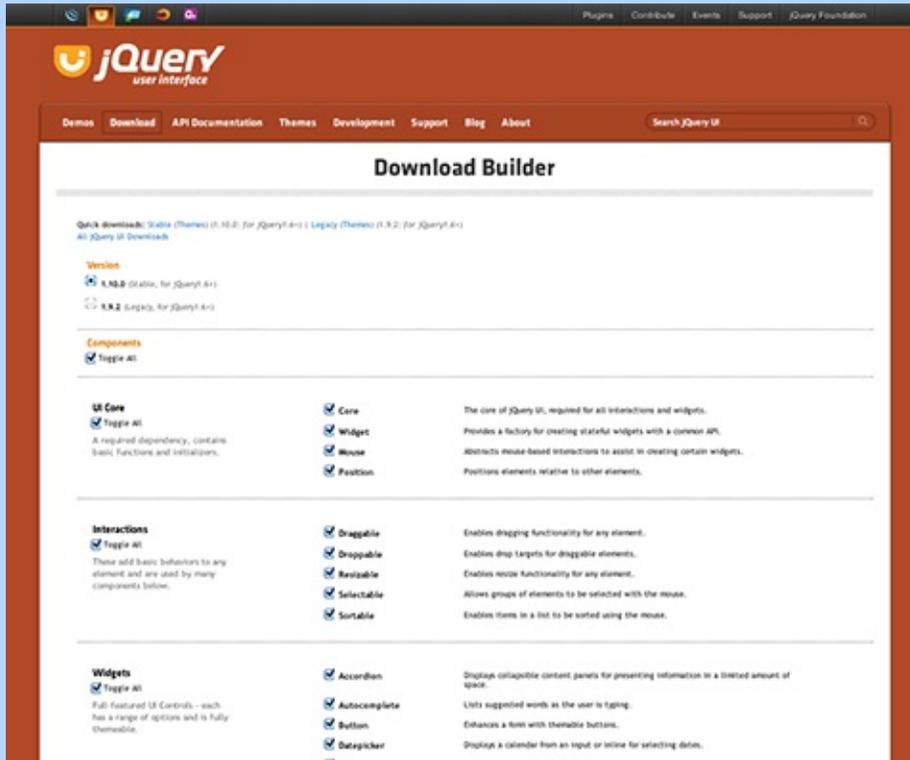
## Build your custom jQuery UI download

Once you have a basic understanding of what jQuery UI is and what it does, you're ready to try it out! It's time to head over to the [Download Builder](#) on the jQuery UI website to download a copy of jQuery UI. jQuery UI's download builder allows you to choose the components you would like to download and get a custom version of the library for your project. There are 3 easy steps to building your custom jQuery UI download:

### Step 1: Choose which components you need

The main column of the Download Builder lists all of the javascript components in jQuery UI categorized into groups: core, interactions, widgets, and effects. Some components in jQuery UI depend on other components. Just check the boxes for the widgets you'd like to download and any required dependencies will automatically be checked as well. The components you select will all be combined

into a custom jQuery UI javascript file.



## Step 2: Select a theme (or roll your own custom theme)

In the right column of the download builder, you'll find a field where you can choose from a number of pre-designed themes for your jQuery UI widgets. You can either choose from the various themes we provide, or you can design your own custom theme using ThemeRoller (more on that later).

**Advanced Theme Settings:** *The theme section of the download builder also offers some advanced configuration settings for your theme. If you plan to use multiple themes on a single page, these fields will come in handy. If you plan to only use one theme on a page, you can skip these settings entirely.*

## Step 3: Choose a version of jQuery UI

The last step in the download builder is to select a version number. This is a very important step because jQuery UI versions are designed to work with specific versions of jQuery. The current versions are:

- jQuery UI 1.10.0 - Requires jQuery 1.6+
- jQuery UI 1.9.2 - Requires jQuery 1.6+

## Click Download!

You're finished with the download builder! Click the download button and you'll get a customized zip file containing everything you selected.

## After downloading: Intro to using jQuery UI

Once you've downloaded jQuery UI, you'll get a zip containing the following files:

- `/css/`
- `/development-bundle/`
- `/js/`
- `index.html`

## Basic overview: using jQuery UI on a web page

Open up `index.html` in a text editor and you'll see that it links to a few dependencies: your theme, jQuery, and jQuery UI. Generally, you'll need to include these 3 files on any page to use jQuery UI widgets and interactions:

```
1 <link type="text/css" href="css/themename/jqu
2 <script type="text/javascript" src="js/jquery
3 <script type="text/javascript" src="js/jquery
```

Once you've included the necessary files, you can add some jQuery widgets to your page. For example, to make a datepicker widget, you'll add a text input element to your page and then call `.datepicker();` on it. Like this:

## HTML:

```
1 <input type="text" name="date" id="date">
```

## JS:

```
1 $('#date').datepicker();
```



## That's it!

For demos of all of the jQuery UI widgets and interactions. Check out the demos section of the jQuery UI website.

## Customizing jQuery UI to your needs

jQuery UI allows you to customize it in several ways. You've already seen how the download builder allows you to customize your copy of jQuery UI to include only the portions you want, but there are additional ways to customize that code to your implementation.

## jQuery UI basics: using options

Each plugin in jQuery UI has a default configuration which is catered to the most basic and common use case. But if you want a plugin to behave different from its default configuration, you can override each of its default settings using "options". Options are a set of properties passed into a jQuery UI widget as an argument.

For example, the slider widget has an option for orientation, which allows you to specify whether the slider should be horizontal or vertical. To set this option for a slider on your page, you just pass it in as an argument, like this:

```
1 | $('#mySliderDiv').slider({
2 |     orientation: 'vertical'
3 | });
```

You can pass as many different options as you'd like by following each one with a comma (except the last one):

```
1 | $('#mySliderDiv').slider({
2 |     orientation: 'vertical',
3 |     min: 0,
4 |     max: 150,
5 |     value: 50
6 | });
```

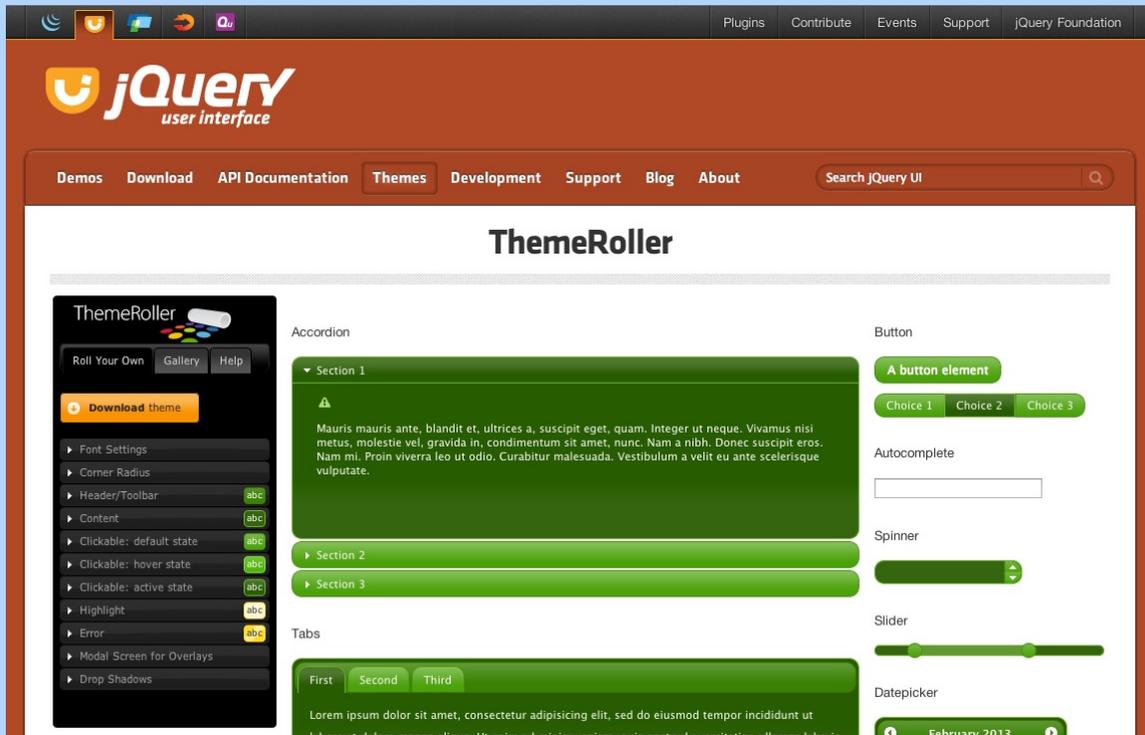
Just remember to surround your options with curly brackets `{ }`, and you're well on your way. Of course, the example above barely touches on what you can do with jQuery UI. To get detailed information on the entire set of jQuery UI widgets, visit the [jQuery UI documentation](#).

## Visual customization: Designing a jQuery UI theme

If you want to design your own theme, jQuery UI has a very slick application for just that purpose. It's called ThemeRoller, and you can always get to it by either clicking the "Themes" link in the jQuery UI navigation, or simply going to [jQuery UI Themeroller](#).

ThemeRoller provides a custom interface for designing all of the elements used by jQuery UI widgets. As you tweak the "levers" in the left column, the widgets on the right will reflect your design. The

Gallery tab of ThemeRoller offers a number of pre-designed themes (the same ones offered by the download builder) that you can tweak or download as they are.



## Downloading your theme

When you click the "Download theme" button in ThemeRoller, you'll be directed to the Download Builder and your custom theme will be auto-selected in the Theme dropdown menu. You can configure your download package further from there. Once you download, you'll see that the example.html page is styled using your custom theme.

**Quick tip:** *If you ever need to edit your theme, simply open the CSS file and find on line 43 where it says "To view and modify this theme, visit ...". That url will open the theme in ThemeRoller for editing.*

## Support: Where can I get help?

JQuery UI user and developer resources are kept up-to-date at the [Support Center](#).

## Developers Wanted!

Want to join the jQuery UI team? We'd love your help! Visit the [UI Development Center](#) for details on how to get involved.

# How jQuery UI Works

jQuery UI contains many widgets that maintain [state](#) and therefore may have a slightly different usage pattern than typical jQuery plugins you are already used to. While the initialization is the same as most jQuery plugins, jQuery UI's widgets are built on top of the [Widget Factory](#) which provides the same general API to all of them. So if you learn how to use one, then you'll know how to use all of them! This document will walk you through the common functionality, using the [progressbar](#) widget for the code examples.

## Initialization

In order to track the state of the widget, we must introduce a full life cycle for the widget. The life cycle starts when the widget is initialized. To initialize a widget, we simply call the plugin on one or more elements.

```
1 | $( "#elem" ).progressbar();
```

This will initialize each element in the jQuery object, in this case the element with an id of "elem". Because we called the `progressbar()` method with no parameters, the widget is initialized with its default options. We can pass a set of options during initialization in order to override the default options.

```
1 | $( "#elem" ).progressbar({ value: 20 });
```

We can pass as many or as few options as we want during initialization. Any options that we don't pass will just use their default values.

The options are part of the widget's state, so we can set options after initialization as well. We'll see this later with the `option` method.

## Methods

Now that the widget is initialized, we can query its state or perform actions on the widget. All actions after initialization take the form of a method call. To call a method on a widget, we pass the name of the method to the jQuery plugin. For example, to call the `value` method on our progressbar widget, we would use:

```
1 | $( "#elem" ).progressbar( "value" );
```

If the method accepts parameters, we can pass them after the method name. For example, to pass the parameter `40` to the `value` method, we can use:

```
1 | $( "#elem" ).progressbar( "value", 40 );
```

Just like other methods in jQuery, most widget methods return the jQuery object for chaining.

```
1 | $( "#elem" )  
2 |   .progressbar( "value", 90 )  
3 |   .addClass( "almost-done" );
```

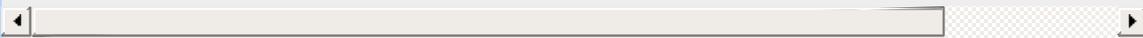
## Common Methods

Each widget will have its own set of methods based on the functionality that the widget provides. However, there are a few methods that exist on all widgets.

### option

As we mentioned earlier, we can change options after initialization through the `option` method. For example, we can change the progressbar's value to 30 by calling the `option` method.

```
1 | $( "#elem" ).progressbar( "option", "value"
```



Note that this is different from the previous example where we were calling the `value` method. In this example, we're calling the `option` method and saying that we want to change the value option to 30.

We can also get the current value for an option.

```
1 | $( "#elem" ).progressbar( "option", "value"
```



In addition, we can update multiple options at once by passing an object to the `option` method.

```
1 | $( "#elem" ).progressbar( "option", {  
2 |     value: 100,  
3 |     disabled: true  
4 | });
```

You may have noticed that the `option` method has the same signature as getters and setters in jQuery core, such as `.css()` and `.attr()`. The only difference is that you have to pass the string "option" as the first parameter.

## disable

As you might guess, the `disable` method disables the widget. In the case of progressbar, this changes the styling to make the progressbar look disabled.

```
1 | $( "#elem" ).progressbar( "disable" );
```

Calling the `disable` method is equivalent to setting the `disabled`

option to `true`.

## enable

The `enable` method is the opposite of the `disable` method.

```
1 | $( "#elem" ).progressbar( "enable" );
```

Calling the `enable` method is equivalent to setting the `disabled` option to `false`.

## destroy

If you no longer need the widget, you can destroy it and return back to the original markup. This ends the life cycle of the widget.

```
1 | $( "#elem" ).progressbar( "destroy" );
```

Once you destroy a widget, you can no longer call any methods on it unless you initialize the widget again. If you're removing the element, either directly via `.remove()` or by modifying an ancestor with `.html()` or `.empty()`, the widget will automatically destroy itself.

## widget

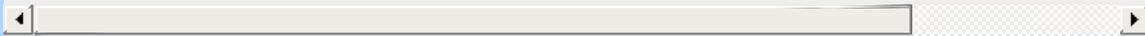
Some widgets generate wrapper elements, or elements disconnected from the original element. In these cases, the `widget` method will return the generated element. In cases like the progressbar, where there is no generated wrapper, the `widget` method returns the original element.

```
1 | $( "#elem" ).progressbar( "widget" );
```

## Events

All widgets have events associated with their various behaviors to notify you when the state is changing. For most widgets, when the events are triggered, the names are prefixed with the widget name. For example, we can bind to progressbar's change event which is triggered whenever the value changes.

```
1 $( "#elem" ).bind( "progressbarchange", function() {
2     alert( "The value has changed!" );
3 });
```



Each event has a corresponding callback, which is exposed as an option. We can hook into progressbar's `change` callback instead of binding to the `progressbarchange` event, if we wanted to.

```
1 $( "#elem" ).progressbar({
2     change: function() {
3         alert( "The value has changed!" );
4     }
5 });
```

## Common Events

While most events will be widget specific, all widgets have a `create` event. This event will be triggered immediately after the widget is created.

# Theming jQuery UI

All jQuery UI plugins are designed to allow a developer to seamlessly integrate UI widgets into the look and feel of their site or application. Each plugin is styled with CSS and contains two layers of style information: standard [jQuery UI CSS Framework](#) styles and plugin-specific styles.

The jQuery UI CSS Framework provide semantic presentation classes to indicate the role of an element within a widget such as a header, content area, or clickable region. These are applied consistently across all widgets so a clickable tab, accordion or button will all have the same `ui-state-default` class applied to indicate that it is clickable. When a user mouses over one of these elements, this class is changed to `ui-state-hover`, then `ui-state-active` when selected. This level of class consistency makes it easy to ensure that all elements with a similar role or interaction state will look the same across all widgets.

The CSS Framework styles are encapsulated in a single file called `ui.theme.css` and this is the file modified by the [ThemeRoller](#) application. Framework styles only include attributes that affect the look and feel (primarily color, background images and icons) so these are 'safe' styles that will not affect functionality of individual plugins. This separation means that a developer can create a custom look and feel by modifying the colors and images in `theme.css` file and know that as future plugins or bug fixes become available, these should work with the theme without modification.

Since the framework styles only cover look and feel, plugin specific stylesheets are included that contain all the additional structural style rules required to make the widget functional, such as dimensions, padding, margins, positioning and floats. Stylesheets for each plugin are located in the `themes/base` folder of the download and are named to match the plugin such as `"jquery.ui.accordion.css"`. These styles must be carefully edited because they work in conjunction with the scripting and provide overrides of framework styles as needed.

We encourage all developers creating jQuery plugins to leverage

the jQuery UI CSS Framework because it will make it much easier for end users to theme and use your plugin.

## Getting started

There are three general approaches to theming jQuery UI plugins:

- **Download a ThemeRoller theme:** The easiest way to build a theme is to use the [Themeroller](#) to generate and download a theme. This app will create a new `ui.theme.css` file and an images directory containing all necessary background images and icon sprites which can simply be dropped into your project. This approach will be the easiest to create and maintain but limits customization to the options provided in ThemeRoller.
- **Modify the CSS files:** To get a bit more control over the look and feel, you may choose to start with the default theme (Smoothness) or a ThemeRoller-generated theme and then adjust the `ui.theme.css` file or any of the individual plugin stylesheets. For example, you could easily tweak the corner radius for all buttons to be different than the rest of the UI components or change the path for the icon sprite to use a custom set. With a bit of style scoping, you can even use multiple themes together in a single UI. To keep maintenance simple, restricting changes to just the `ui.theme.css` file and images is recommended.
- **Write completely custom CSS:** For the greatest amount of control, the CSS for each plugin can be written from scratch without using the framework classes or plugin-specific stylesheet. This may be necessary if the desired look and feel can't be achieved by modifying the CSS or if highly customized markup is used. This approach requires deep expertise in CSS and will require manual updates for future plugins.

## Using ThemeRoller and Themes

- [ThemeRoller application](#)
- [ThemeRoller documentation](#) - How to use the ThemeRoller application

## jQuery UI CSS Framework & Custom themes

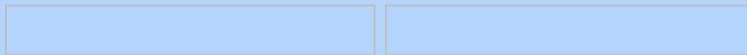
- [jQuery UI CSS Framework documentation](#) - Explore the full class API and icon set
- [How to create custom themes](#) - Tutorial for creating a theme from scratch
- [Using jQuery UI ThemeRoller](#)
- [jQuery UI CSS Framework API](#)
- [How To Write a Theme](#)

# Using jQuery UI ThemeRoller



## About ThemeRoller

ThemeRoller is a web app that offers a fun and intuitive interface for designing and downloading custom themes for jQuery UI. You can find ThemeRoller in the "Themes" section of the jQuery UI site, or by following this link: [jQuery UI ThemeRoller](#)



## The ThemeRoller Interface

The interface for ThemeRoller is categorized into panels for global font and corner radius settings, widget container styles, and interaction states for clickable elements, and various styles for overlays and shadows. These panels allow configuration of various CSS properties such as font size, color, and weight, background color and texture, border color, text color, icon color, corner radius, and more!

## The Theme Gallery: Pre-Rolled Themes

ThemeRoller themes can be viewed via permalink URLs, and it includes a gallery of pre-designed themes to choose from. The theme gallery is accessible through the tab strip located at the top of the application interface. From the gallery, you can preview and download themes, or even choose to tweak a theme further in the "Roll Your Own" tab.

## Downloading Themes

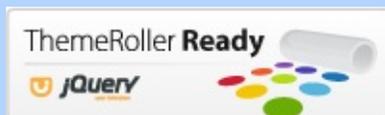
When you're done designing a theme, you can download it for use in your projects. ThemeRoller has a "Download Theme" button at the top which will generate a ZIP file containing all of theme assets. Images included in your download will be generated to your specifications and saved as high-quality PNG files.

Your theme will include images and CSS that make up a customized version of the jQuery UI CSS Framework including images and CSS for all of our plugins.

## Installing downloaded themes into your project

Once the theme has been downloaded and unzipped, you will see a folder named `themes`. This folder contains the CSS and images needed for your theme. Copy the theme directory into your project and link to the `themes/all.css` file from your pages.

## Building Custom "ThemeRoller-Ready" Components



ThemeRoller generates a customized version of the jQuery UI CSS Framework for developing your own ThemeRoller-Ready jQuery components. The classes generated by this framework are designed to accommodate common user interface design situations and include states, icons, and various helper classes as well.

For information on developing with the jQuery UI CSS Framework, visit our [Theming API documentation](#).

## ThemeRoller Links

- [Tutorial: Develop Your Own jQuery "ThemeRoller-Ready" Components, Filament Group](#)
- [Introducing ThemeRoller: Design & Download Custom Themes for jQuery UI, Filament Group](#)

- [Videos about Design & Download Custom Themes for jQuery UI, Filament Group](#)

## Credits

ThemeRoller was designed and developed for jQuery UI by [Filament Group, Inc](#), of Boston, MA.

# jQuery UI CSS Framework API

[Back to jQuery UI Theming](#)

## The jQuery UI CSS Framework

jQuery UI includes a robust CSS Framework designed for building custom jQuery widgets. The framework includes classes that cover a wide array of common user interface needs, and can be manipulated using jQuery UI ThemeRoller. By building your UI components using the jQuery UI CSS Framework, you will be adopting shared markup conventions and allowing for ease of code integration across the plugin community at large.

## Framework Classes

The following CSS classes are split between `ui.core.css` and `ui.theme.css`, depending on whether styles are fixed and structural, or themeable (colors, fonts, backgrounds, etc) respectively. These classes are designed to be applied to User Interface elements to achieve visual consistency across an application and allow components to be themeable by jQuery UI ThemeRoller.

## Layout Helpers

- `.ui-helper-hidden`: Applies `display: none` to elements.
- `.ui-helper-hidden-accessible`: Applies accessible hiding to elements (via `abs` positioning off the page)
- `.ui-helper-reset`: A basic style reset for UI elements. Resets things such as padding, margins, text-decoration, list-style, etc.
- `.ui-helper-clearfix`: Applies float wrapping properties to parent elements
- `.ui-helper-zfix`: Applies `iframe "fix"` css to `iframe` elements when needed in overlays.

## Widget Containers

- `.ui-widget`: Class to be applied on outer container of all widgets. Applies font family and font size to widget. Also applies same family and 1em font size to child form elements specifically, to combat inheritance issues in Win browsers.
- `.ui-widget-header`: Class to be applied to header containers. Applies header container styles to an element and its child text, links, and icons.
- `.ui-widget-content`: Class to be applied to content containers. Applies content container styles to an element and its child text, links, and icons. (can be applied to parent or sibling of header)

## Interaction States

- `.ui-state-default`: Class to be applied to clickable button-like elements. Applies "clickable default" container styles to an element and its child text, links, and icons.
- `.ui-state-hover`: Class to be applied on mouseover to clickable button-like elements. Applies "clickable hover" container styles to an element and its child text, links, and icons.
- `.ui-state-focus`: Class to be applied on keyboard focus to clickable button-like elements. Applies "clickable hover" container styles to an element and its child text, links, and icons.
- `.ui-state-active`: Class to be applied on mousedown to clickable button-like elements. Applies "clickable active" container styles to an element and its child text, links, and icons.

## Interaction Cues

- `.ui-state-highlight`: Class to be applied to highlighted or selected elements. Applies "highlight" container styles to an element and its child text, links, and icons.
- `.ui-state-error`: Class to be applied to error messaging container elements. Applies "error" container styles to an element and its child text, links, and icons.
- `.ui-state-error-text`: An additional class that applies just the

error text color without background. Can be used on form labels for instance. Also applies error icon color to child icons.

- `.ui-state-disabled`: Applies a dimmed opacity to disabled UI elements. Meant to be added in addition to an already-styled element.
- `.ui-priority-primary`: Class to be applied to a priority-1 button in situations where button hierarchy is needed. Applies bold text.
- `.ui-priority-secondary`: Class to be applied to a priority-2 button in situations where button hierarchy is needed. Applies normal weight text and slight transparency to element.

## Icons

### States and images

- `.ui-icon`: Base class to be applied to an icon element. Sets dimensions to 16px square block, hides inner text, sets background image to "content" state sprite image. **Note:** `.ui-icon` class will be given a different sprite background image depending on its parent container. For example, a `ui-icon` element within a `ui-state-default` container will get colored according to the `ui-state-default`'s icon color.

### Icon types

After declaring a `.ui-icon` class, you can follow up with a second class describing the type of icon you'd like. Icon classes generally follow a syntax of `.ui-icon-{icon type}-{icon sub description}-{direction}`.

For example, a single triangle icon pointing to the right looks like this: `.ui-icon-triangle-1-e`

jQuery UI's [ThemeRoller](#) provides the full set of CSS framework icons in its preview column. Hover over them to see the class name.

## Misc Visuals

## Corner Radius helpers

- `.ui-corner-tl`: Applies corner-radius to top left corner of element.
- `.ui-corner-tr`: Applies corner-radius to top right corner of element.
- `.ui-corner-bl`: Applies corner-radius to bottom left corner of element.
- `.ui-corner-br`: Applies corner-radius to bottom right corner of element.
- `.ui-corner-top`: Applies corner-radius to both top corners of element.
- `.ui-corner-bottom`: Applies corner-radius to both bottom corners of element.
- `.ui-corner-right`: Applies corner-radius to both right corners of element.
- `.ui-corner-left`: Applies corner-radius to both left corners of element.
- `.ui-corner-all`: Applies corner-radius to all 4 corners of element.

## Overlay & Shadow

- `.ui-widget-overlay`: Applies 100% wxh dimensions to an overlay screen, along with background color/texture, and screen opacity.
- `.ui-widget-shadow`: Class to be applied to overlay width, top/left offsets and shadow "thickness". Thickness is applied via padding to all sides of a shadow that is set to match the dimensions of the overlay element. Offsets are applied via top and left margins (can be positive or negative).

# How To Write a Theme

## File Structure

Themes are meant to be structured in a specific manner (in order to increase their ease of use). The general file directory structure is as follows:

- `themename/` - Your theme should be completely contained within a single directory (named the same as your theme).
- `themename/themename.css` - This is your base CSS file. This file will be included on every page that uses your theme, regardless of which plugins are being used. (This file should be very lightweight and only include the essentials.)
- `themename/themename.pluginname.css` - You'll need one CSS file for each plugin that you support. The name of the plugin should be included directly in the filename (for example, if you themed the tabs plugin, you would have: `themename.tabs.js`).
- `themename/img.png` - Your theme can include images, if need be. They can be named whatever you'd like (there is no, particular, naming convention for these).

For examples of how theme file structures are done, feel free to look through the only completed theme so far: <http://jquery-ui.googlecode.com/svn/trunk/themes/flora/>

## Styling

Writing styles for themes is rather simplistic - which is good, as it allows for a great amount of flexibility.

All themes should have a base CSS class to work off of. This main class allows the user to enable, and switch between, themes. Your root class should be of the format `.ui-themename`. And it will be used like so, within a user's HTML file:

```
1 <html>
```

```
2 <head>
3   <title>My Site</title>
4   <link rel="stylesheet" href="themename/themename.css" />
5   <link rel="stylesheet" href="othertheme/othertheme.css" />
6   <link rel="stylesheet" href="othertheme/othertheme.d" />
7 </head>
8 <body class="ui-themename">
9   ...
10
11   <div class="ui-othertheme">
12     <div class="ui-dialog">... modal dialog ...
13   </div>
14 </body>
15 </html>
```

In the above example, a couple important things are occurring:

- We're loading two themes into a document at the same time.
- The whole page, and all of its contents, are themed according to themename's styling.
- However, the div with the class of ui-othertheme, and everything inside of it (including the modal dialog), is themed according to othertheme's styling rules.

If we were to take a look inside `themename.css` we might see something like the following:

```
1 body.ui-themename { background:#111; color:sr
2 .ui-themename a, a.ui-themename { color:#68D;
3 .ui-themename a:visited, a.ui-themename:visit
4 .ui-themename a:hover, a.ui-themename:hover {
```

If you'll notice, the `themename.css` file only contains styling information that is globally applicable (nothing plugin-specific is

contained within here). This will be true for all cases where a theme is built. Don't worry about a theme taking up multiple files - that will be simplified within the build and download process.

# Widget Factory

The jQuery UI Widget Factory is an extensible base on which all of jQuery UI's widgets are built. Using the widget factory to build a plugin provides conveniences for state management, as well as conventions for common tasks like exposing plugin methods and changing options after instantiation.

- [Why Use the Widget Factory](#)
- [How To Use the Widget Factory](#)

# Why Use the Widget Factory

Writing jQuery plugins is as simple as adding a method to `jQuery.prototype` (more commonly seen as `$.fn`) and following some simple conventions like returning `this` for chainability. So why does the widget factory exist? And why is it hundreds of lines of code?

In this document, we'll walk through the benefits of the widget factory and find out when and why it makes sense to use it.

## Stateless vs. Stateful Plugins

Most jQuery plugins are stateless; they perform some action and their job is done. For example, if you set the text of an element using `.text( "hello" )`, there is no setup phase and the result is always the same. For these types of plugins, it makes sense to just extend jQuery's prototype.

However, some plugins are stateful; they have full life cycles, maintain state, and react to changes. These plugins require a lot of code dedicated to initialization and state management (and sometimes destruction). This results in a lot of boilerplate for building stateful plugins. Even worse, each plugin author may manage life cycles and state differently, resulting in different API styles for different plugins. The widget factory aims to solve both problems, removing the boilerplate and creating a consistent API across plugins.

## Consistent API

The widget factory defines how to create and destroy widgets, get and set options, invoke methods, and listen to events triggered by the widget. By using the widget factory to build your stateful plugins, you are automatically conforming to a defined standard, making it easier for new users to start using your plugins. In addition to defining the interface, the widget factory also implements much of this functionality for you. If you're not familiar with the API provided by the widget factory, you should read [How jQuery UI Works](#).

## Setting Options on Initialization

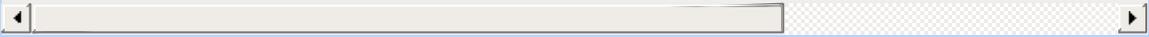
Whenever you build a plugin that accepts options, you should define defaults for as many options as possible, then merge the user-provided options with the defaults on initialization. It's also a good idea to expose the defaults so that users can even change the default values. A common pattern in jQuery plugins looks like this:

```
1   $.fn.plugin = function( options ) {
2       options = $.extend( {}, $.fn.plugin.defa
3       // plugin logic goes here
4   };
5
6   $.fn.plugin.defaults = {
7       param1: "foo",
8       param2: "bar",
9       param3: "baz"
10  };
```

The widget factory provides this functionality and even takes it a bit further. Let's see what this looks like with the widget factory.

```
1   $.widget( "ns.plugin", {
2
3       // default options
4       options: {
5           param1: "foo",
6           param2: "bar",
7           param3: "baz"
8       },
9
10      _create: function() {
11          // options are already merged and stor
12          // plugin logic goes here
```

```
13     }  
14     });
```



# How To Use the Widget Factory

To start, we'll create a progress bar that just lets us set the progress once. As we can see below, this is done by calling `jQuery.widget` with two parameters: the name of the plugin to create and an object literal containing functions to support our plugin. When our plugin gets called, it will create a new plugin instance and all functions will be executed within the context of that instance. This is different from a standard jQuery plugin in two important ways. First, the context is an object, not a DOM element. Second, the context is always a single object, never a collection.

```
1     $.widget( "custom.progressbar", {
2         _create: function() {
3             var progress = this.options.value
4             this.element
5                 .addClass( "progressbar" )
6                 .text( progress );
7         }
8     });
```

The name of the plugin must contain a namespace, in this case we've used the `custom` namespace. There is currently a limitation that exactly one namespace must be used. We can also see that the widget factory has provided two properties for us. `this.element` is a jQuery object containing exactly one element. If our plugin is called on a jQuery object containing multiple elements, a separate plugin instance will be created for each element, and each instance will have its own `this.element`. The second property, `this.options`, is a hash containing key/value pairs for all of our plugin's options. These options can be passed to our plugin as shown here.

```
1     $( "<div></div>" )
2         .appendTo( "body" )
3         .progressbar({ value: 20 });
```

When we call `jQuery.widget` it extends jQuery by adding a function to `jQuery.fn` (the system for creating a standard plugin). The name of the function it adds is based on the name you pass to `jQuery.widget`, without the namespace; in our case “progressbar”. The options passed to our plugin are the values that get set in `this.options` inside of our plugin instance. As shown below, we can specify default values for any of our options. When designing your API, you should figure out the most common use case for your plugin so that you can set appropriate default values and make all options truly optional.

```
1     $.widget( "custom.progressbar", {
2         // default options
3         options: {
4             value: 0
5         },
6         _create: function() {
7             var progress = this.options.value;
8             this.element
9                 .addClass( "progressbar" )
10                .text( progress );
11        }
12    });
```

## Calling plugin methods

Now that we can initialize our progress bar, we'll add the ability to perform actions by calling methods on our plugin instance. To define a plugin method, we just include the function in the object literal that we pass to `jQuery.widget`. We can also define “private” methods by prepending an underscore to the function name.

```
1     $.widget( "custom.progressbar", {
2         options: {
```

```

3         value: 0
4     },
5     _create: function() {
6         var progress = this.options.value +
7         this.element
8             .addClass( "progressbar" )
9             .text( progress );
10    },
11    // create a public method
12    value: function( value ) {
13        // no value passed, act as a getter
14        if ( value === undefined ) {
15            return this.options.value;
16        // value passed, act as a setter
17        } else {
18            this.options.value = this._constrain(
19            var progress = this.options.value +
20            this.element.text( progress );
21        }
22    },
23    // create a private method
24    _constrain: function( value ) {
25        if ( value > 100 ) {
26            value = 100;
27        }
28        if ( value < 0 ) {
29            value = 0;
30        }
31        return value;
32    }
33    });

```

To call a method on a plugin instance, you pass the name of the method to the jQuery plugin. If you are calling a method that accepts parameters, you simply pass those parameters after the method name.

**Note:** Executing methods by passing the method name to the same jQuery function that was used to initialize the plugin may seem odd. This is done to prevent pollution of the jQuery namespace while maintaining the ability to chain method calls. Later in this article we'll see alternative uses that may feel more natural.

```
1   var bar = $( "<div></div>" )
2       .appendTo( "body" )
3       .progressbar({ value: 20 });
4
5   // get the current value
6   alert( bar.progressbar( "value" ) );
7
8   // update the value
9   bar.progressbar( "value", 50 );
10
11  // get the current value again
12  alert( bar.progressbar( "value" ) );
```

## Working with options

One of the methods that are automatically available to our plugin is the `option` method. The `option` method allows you to get and set options after initialization. This method works exactly like jQuery's `css` and `attr` methods: you can pass just a name to use it as a getter, a name and value to use it as a single setter, or a hash of name/value pairs to set multiple values. When used as a getter, the plugin will return the current value of the option that corresponds to the name that was passed in. When used as a setter, the plugin's `_setOption` method will be called for each option that is being set. We can specify a `_setOption` method in our plugin to react to option changes. For actions to perform independent of the number of options changed, we can override `_setOptions`.

```
1   $.widget( "custom.progressbar", {
2       options: {
```

```

3         value: 0
4     },
5     _create: function() {
6         this.element.addClass( "progressbar"
7         this.refresh();
8     },
9     _setOption: function( key, value ) {
10        if ( key === "value" ) {
11            value = this._constrain( value );
12        }
13        this._super( key, value );
14    },
15    _setOptions: function( options ) {
16        this._super( options );
17        this.refresh();
18    },
19    refresh: function() {
20        var progress = this.options.value +
21        this.element.text( progress );
22    },
23    _constrain: function( value ) {
24        if ( value > 100 ) {
25            value = 100;
26        }
27        if ( value < 0 ) {
28            value = 0;
29        }
30        return value;
31    }
32    });

```

## Adding callbacks

One of the easiest ways to make your plugin extensible is to add callbacks so users can react when the state of your plugin changes. We can see below how to add a callback to our progress bar to

signify when the progress has reached 100%. The `_trigger` method takes three parameters: the name of the callback, a jQuery event object that initiated the callback, and a hash of data relevant to the event. The callback name is the only required parameter, but the others can be very useful for users who want to implement custom functionality on top of your plugin. For example, if we were building a draggable plugin, we could pass the mousemove event when triggering a drag callback; this would allow users to react to the drag based on the x/y coordinates provided by the event object. Note that the original event passed to `_trigger` must be a jQuery event, not a native browser event.

```
1     $.widget( "custom.progressbar", {
2         options: {
3             value: 0
4         },
5         _create: function() {
6             this.element.addClass( "progress" );
7             this.refresh();
8         },
9         _setOption: function( key, value ) {
10            if ( key === "value" ) {
11                value = this._constrain( value );
12            }
13            this._super( key, value );
14        },
15        _setOptions: function( options ) {
16            this._super( options );
17            this.refresh();
18        },
19        refresh: function() {
20            var progress = this.options.value;
21            this.element.text( progress );
22            if ( this.options.value == 100 ) {
23                this._trigger( "complete", this );
24            }
25        },
```

```

26         _constrain: function( value ) {
27             if ( value > 100 ) {
28                 value = 100;
29             }
30             if ( value < 0 ) {
31                 value = 0;
32             }
33             return value;
34         }
35     });

```

Callback functions are essentially just additional options, so you can get and set them just like any other option. Whenever a callback is executed, a corresponding event is triggered as well. The event type is determined by concatenating the plugin name and the callback name. The callback and event both receive the same two parameters: an event object and a hash of data relevant to the event, as we'll see below. Your plugin may have functionality that you want to allow the user to prevent. The best way to support this is by creating cancelable callbacks. User's can cancel a callback, or its associated event, the same way they cancel any native event, by calling `event.preventDefault()` or returning false. If the user cancels the callback, the `_trigger` method will return false so you can implement the appropriate functionality within your plugin.

```

1     var bar = $( "<div></div>" )
2         .appendTo( "body" )
3         .progressbar({
4             complete: function( event, data
5                 alert( "Callbacks are great!"
6             )
7         })
8         .bind( "progressbarcomplete", functi
9             alert( "Events bubble and suppor
10            alert( "The progress bar value i

```

```
11         });  
12  
13         bar.progressBar( "option", "value", 100 );
```

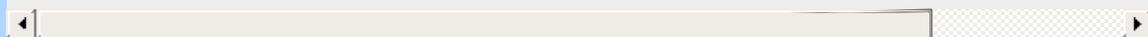


## Looking under the hood

Now that we've seen how to build a plugin using the widget factory, let's take a look at how it actually works. When you call `jQuery.widget`, it creates a constructor for your plugin and sets the object literal that you pass in as the prototype for your plugin instances. All of the functionality that automatically gets added to your plugin comes from a base widget prototype, which is defined as `jQuery.widget.prototype`. When a plugin instance is created, it is stored on the original DOM element using `jQuery.data`, with the plugin name as the key.

Because the plugin instance is directly linked to the DOM element, you can access the plugin instance directly instead of going through the exposed plugin method if you want. This will allow you to call methods directly on the plugin instance instead of passing method names as strings and will also give you direct access to the plugin's properties.

```
1      var bar = $( "<div></div>" )
2          .appendTo( "body" )
3          .progressbar()
4          .data( "progressbar" );
5
6      // call a method directly on the plugin
7      bar.option( "value", 50 );
8
9      // access properties on the plugin instance
10     alert( bar.options.value );
```



You can also create an instance without going through the plugin method, by calling the constructor directly, with the options and element arguments:

```
1      var bar = $.custom.progressbar( {}, $( "<
```

```
3 // same result as before
4 alert( bar.options.value );
```

## Extending a plugin's prototype

One of the biggest benefits of having a constructor and prototype for a plugin is the ease of extending the plugin. By adding or modifying methods on the plugin's prototype, we can modify the behavior of all instances of our plugin. For example, if we wanted to add a method to our progress bar to reset the progress to 0% we could add this method to the prototype and it would instantly be available to be called on any plugin instance.

```
1 $.custom.progressbar.prototype.reset = fu
2     this._setOption( "value", 0 );
3     };
```

## Cleaning up

In some cases, it will make sense to allow users to apply and then later unapply your plugin. You can accomplish this via the `_destroy` method. Within the `_destroy` method, you should undo anything your plugin may have done during initialization or later use. `_destroy` is called by the `destroy` method, which is automatically called if the element that your plugin instance is tied to is removed from the DOM, so this can be used for garbage collection as well. That base `destroy` method also handles some general cleanup operations, like removing the instance reference from the widget's DOM element, unbinding all events in the widget's namespace from the element, and unbinding generally all events that were added using `_bind`.

```
1 $.widget( "custom.progressbar", {
2     options: {
```

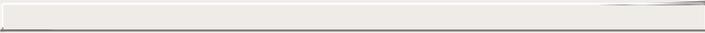
```

3         value: 0
4     },
5     _create: function() {
6         this.element.addClass( "progressbar"
7         this.refresh();
8     },
9     _setOption: function( key, value ) {
10        if ( key === "value" ) {
11            value = this._constrain( value );
12        }
13        this._super( key, value );
14    },
15    _setOptions: function( options ) {
16        this._super( options );
17        this.refresh();
18    },
19    refresh: function() {
20        var progress = this.options.value +
21        this.element.text( progress );
22        if ( this.options.value == 100 ) {
23            this._trigger( "complete", null
24        }
25    },
26    _constrain: function( value ) {
27        if ( value > 100 ) {
28            value = 100;
29        }
30        if ( value < 0 ) {
31            value = 0;
32        }
33        return value;
34    },
35    _destroy: function() {
36        this.element
37            .removeClass( "progressbar" )
38            .text( "" );
39    }

```

40

});



## Closing comments

The widget factory is only one way of creating stateful plugins. There are a few different models that can be used and each have their own advantages and disadvantages. The widget factory solves lots of common problems for you and can greatly improve productivity, it also greatly improves code reuse, making it a great fit for jQuery UI as well as many other stateful plugins.

You may have noticed that in this article we used the `custom` namespace. The `ui` namespace is reserved for official jQuery UI plugins. When building your own plugins, you should create your own namespace. This makes it clear where the plugin came from and if it is part of a larger collection.

# jQuery Mobile

jQuery Mobile is the easiest way to build sites and apps that are accessible on all popular smartphone, tablet and desktop devices. This framework provides a set of touch-friendly UI widgets and an AJAX-powered navigation system to support animated page transitions.

- [Getting Started with jQuery Mobile](#)
- [Creating a custom theme with ThemeRoller](#)

# Getting Started with jQuery Mobile

jQuery Mobile provides a set of touch-friendly UI widgets and an AJAX-powered navigation system to support animated page transitions. This guide will show you how you can build your first jQuery Mobile application.

## Create a basic page template

To get started, you can simply paste the template below in your favorite text editor, save and open the document in a browser.

In the `head` of this template, a meta `viewport` tag sets the screen width to the pixel width of the device. References to jQuery, jQuery Mobile and the mobile theme stylesheet from the CDN add all the styles and scripts. jQuery Mobile 1.2 (1.2.0) works with versions of jQuery core from 1.7.0 to 1.8.2.

In the `body`, a div with a `data-role` of `page` is the wrapper used to delineate a page. A header bar (`data-role="header"`), a content region (`data-role="content"`) and a footer bar (`data-role="footer"`) are added inside to create a basic page (all three are optional). These `data-` attributes are HTML5 attributes used throughout jQuery Mobile to transform basic markup into an enhanced and styled widget.

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title>My Page</title>
5      <meta name="viewport" content="width=device-width,initial-scale=1">
6      <link rel="stylesheet" href="http://code.jquery.com/mobile/1.2.0/jquery.mobile-1.2.0.min.css">
7      <script src="http://code.jquery.com/jquery-1.7.2.min.js"></script>
8      <script src="http://code.jquery.com/mobile/1.2.0/jquery.mobile-1.2.0.min.js"></script>
9    </head>
10   <body>
11
12     <div data-role="page">
13       <div data-role="header">
14         <h1>My Title</h1>
15       </div><!-- /header -->
16
17       <div data-role="content">
18         <p>Hello world</p>
19       </div><!-- /content -->
```

```
20
21     <div data-role="footer">
22         <h4>My Footer</h4>
23     </div><!-- /header -->
24
25 </div><!-- /page -->
26
27 </body>
28 </html>
```

## Add content

The next step is to add content inside the content container. Any standard HTML elements - headings, lists, paragraphs, etc can be added. You can write your own custom styles to create custom layouts by adding an additional stylesheet to the `head` after the jQuery Mobile stylesheet.

## Make a listview

jQuery Mobile includes a diverse set of common listviews that are coded as lists with a `data-role="listview"` added. Here is a simple linked list that has a role of `listview`. The `data-inset="true"` attribute makes the listview look like an inset module, while `data-filter="true"` adds a dynamic search filter.

```
1 <ul data-role="listview" data-inset="true" da
2     <li><a href="#">Acura</a></li>
3     <li><a href="#">Audi</a></li>
4     <li><a href="#">BMW</a></li>
5     <li><a href="#">Cadillac</a></li>
6     <li><a href="#">Ferrari</a></li>
7 </ul>
```

## Add a slider

The framework contains a full set of form elements that automatically are enhanced into touch-friendly styled widgets. Here's a slider made with the new HTML5 input type of range, no `data-role` needed. All form elements must always be properly associated with a `label` and the group of form elements be wrapped in a `form` tag.

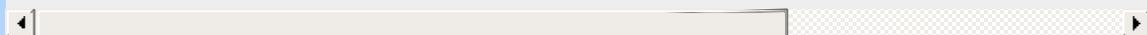
```
1 | <form>
2 |   <label for="slider-0">Input slider:</label>
3 |   <input type="range" name="slider" id="slide
4 | </form>
```



## Make a button

There are a few ways to make buttons. A common one is to turn a link into a button so it's easy to click. Just start with a link and add a `data-role="button"` attribute to it. You can add an icon with the `data-icon` attribute and optionally set its position with the `data-iconpos` attribute.

```
1 | <a href="#" data-role="button" data-icon="sta
```



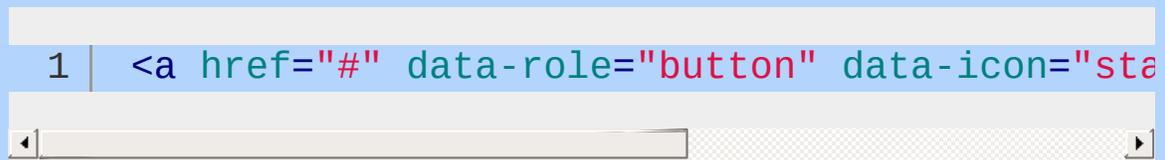
## Choose a theme swatch

jQuery Mobile has a robust theme framework that supports up to 26 sets of toolbar, content and button colors, called a "swatch". You can add a `data-theme="e"` attribute to any of the widgets on this page: page, header, list, input for the slider, or button to turn it yellow. Different swatch letters in default theme from a-e can be used to mix and match swatches.

If you add the theme swatch to the page, all the widgets inside the

content will automatically inherit the theme (headers and footers don't inherit and default to swatch "a").

```
1 | <a href="#" data-role="button" data-icon="sta
```



If you would like to create a custom theme, you can use [ThemeRoller](#) that allows users to create their own theme through an easy to use drag and drop interface. You will then be able to download and use your newly created theme.

## Go forth and build something

This guide has provided you with a basic structure for a jQuery Mobile page and a few enhanced elements. You can explore the full [jQuery Mobile documentation](#) to learn about linking pages, adding animated page transitions, and creating dialogs and popups.

If you're more of the type who prefers actually writing JavaScript to build your apps, and you don't want to use the `data-` attribute configuration system, you can take full control of everything and call plugins directly as these are all standard jQuery plugins built with the UI widget factory. Particularly useful information for such cases can be found in the global configuration, events, and methods sections.

Finally, you can read up on scripting pages, generating dynamic pages, and building PhoneGap apps.

# Creating a custom theme with ThemeRoller

## Theming Overview

jQuery Mobile has a robust theme framework that supports up to 26 sets of toolbar, content and button colors, called a "swatch". The framework comes with five defined themes (swatches 'a' to 'e') which can be used readily, removed or overwritten.

### Default theme swatch mapping for components

If no theme swatch letter is set at all, the framework uses the 'a' swatch (black in the default theme) for headers and footers and the 'c' swatch (light gray in the default theme) for the page content to maximize contrast between the both.

All items in containers inherit the swatch from their parent. Exceptions to this rule are the listdivider in listviews, the header of nested list pages, and the button of split button lists, which all default to 'b' (blue in the default theme). Count bubbles default to 'c' (silver in the default theme).

Note that there is also a swatch named "active" (bright blue in the default theme) which is used to indicate an active selected item. See the Global "Active" state further down this page for further information on the active swatch.

The page loading dialog and error message don't inherit a swatch theme. The loading dialog defaults to swatch 'a' (black in the default theme) and the error message to swatch 'e' (yellow in the default theme). You can configure those defaults globally.

### Themes and swatches

The theme system separates color and texture from structural styles that define things like padding and dimensions. This allows theme colors and textures to be defined once in the stylesheet and to be mixed, matched, and combined to achieve a wide range of visual effects.

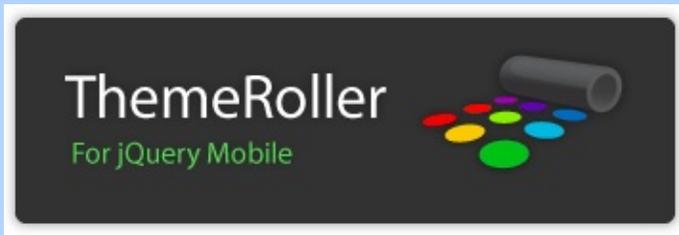
Each theme includes several global settings, including font family,

drop shadows for overlays, and corner radius values for buttons and boxes. In addition, the theme can include multiple color swatches, each with color values for bars, content blocks, buttons and list items, and font text-shadow.

The default theme includes 5 swatches that are given letters (a, b, c, d, e) for quick reference. To make mapping of color swatches consistent across our widgets, we have followed the convention that swatch 'a' is the highest level of visual priority (black in our default theme), 'b' is secondary level (blue) and 'c' is the baseline level (gray) that we use by default in many situations, 'd' for an alternate secondary level and 'e' as an accent swatch. Themes may have additional swatches for accent colors or specific situations. For example, you could add a new theme swatch 'f' that has a red bar and button for use in error situations.

Most theme changes can be done using ThemeRoller, but it is also fairly simple to manually edit the base swatches in the default theme and/or add additional swatches by editing the theme CSS file. Just copy a block of swatch styles, rename the classes with the new swatch letter name, and tweak colors as you see fit.

## Creating a Custom Theme with ThemeRoller



The easiest way to create custom themes is with the ThemeRoller tool. It allows you to build a theme composed of up to 26 swatches, download a the newly created CSS file, and use it in your project.

### Creating the Theme swatches

The [ThemeRoller](#) allows users to create their own theme through an easy to use drag and drop interface. By default, ThemeRoller offers three swatches (a, b and c). You can use the offered default colors, the Adobe kuler colors, or create your own. You will create your theme by dragging the chosen color onto the chosen element in the swatch of your choice. You can add more swatches by pressing the '+' sign near the 'A', 'B', and 'C' tabs, in the left hand-side menu.

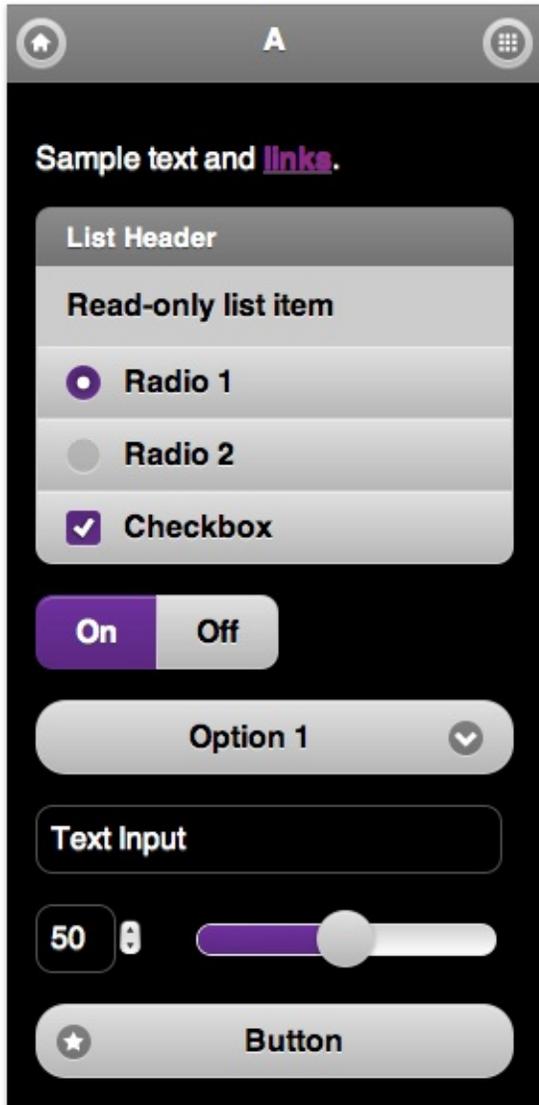
You can further edit your swatch from the menu. For example, you can expand the various element parts and carry out detailed editing. For example, this will allow you to text color, text shadow size, position and color, etc. You can also edit the gradient used on each element.

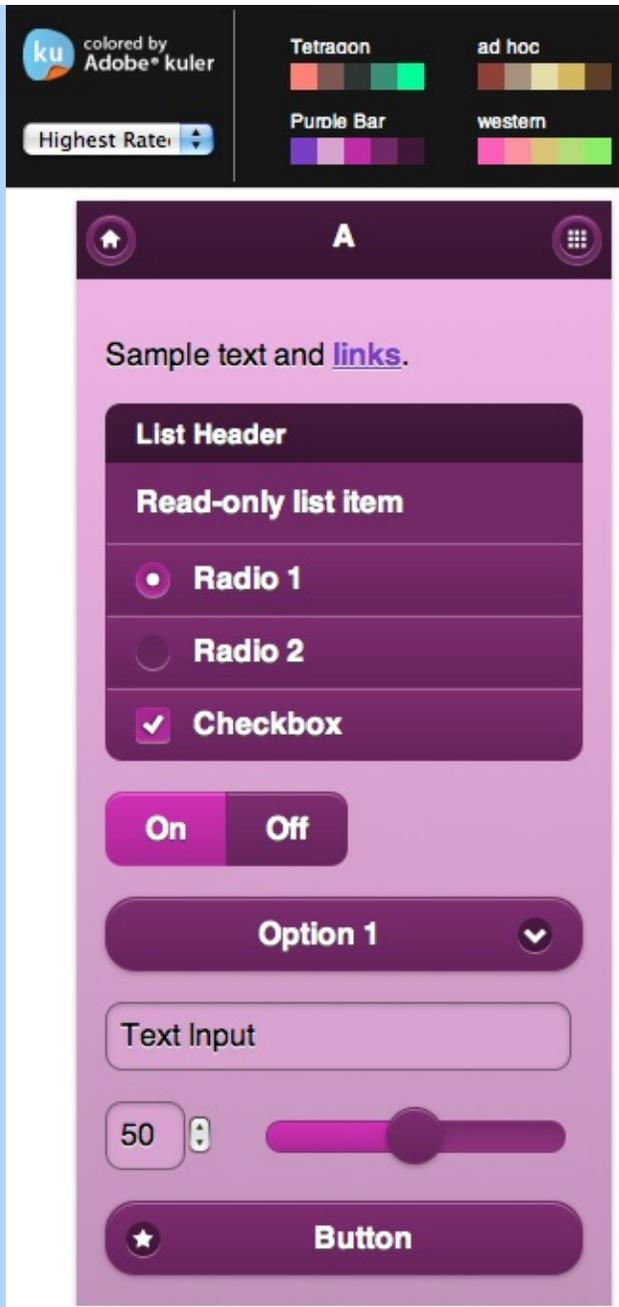
Here are two examples of theme swatches created, the first one with the default colors and with the kuler colors:

Drag a color onto an element below or pick from the



LIGHTNESS  SATURATION





## Downloading the created Theme

Once you are satisfied with the various swatches that you have created in your theme, you can download this theme to be able to start using it in your project. You will simply need to press the 'Download theme zip file button', and enter the name of your theme in the popup window. Then, press the 'Download Zip' button on the download popup window, see below:

## Download Theme

Theme Name

This will generate a Zip file that contains both a compressed (for production) and uncompressed (for editing) version of the theme.

To use your theme, add it to the head of your page before the jquery.mobile.structure file, like this:

```
<!DOCTYPE html>
<html>
<head>

  <title>jQuery Mobile page</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="css/themes/my-custom-theme.css" />
  <link rel="stylesheet" href="http://code.jquery.com/mobile/1.2.0/jquery.mobile.structure-1.2.0.min.css" />
  <script src="http://code.jquery.com/jquery-1.8.2.min.js"></script>
  <script src="http://code.jquery.com/mobile/1.2.0/jquery.mobile-1.2.0.min.js"></script>

</head>
```



Tip: To edit your theme later, use the import feature to paste in the uncompressed theme file

Close

Download Zip

## Using the downloaded Theme

The theme gets downloaded on your local machine as a zip file. This contains an `index.html` file, and a `themes` folder. The `index.html` file is an example of how you can now see your theme. The `themes` folder contains your theme CSS files, and the icons that are used by jQuery Mobile.

To start using your theme, you can either start from the provided `index.html` or start from scratch. As explained in the theme download popup window, all you need is to add your theme to the head of your page before the `jquery.mobile.structure` file, like this:

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4
5     <title>jQuery Mobile page</title>
6     <meta charset="utf-8" />
7     <meta name="viewport" content="width=device-width, initial-scale=1">
8     <link rel="stylesheet" href="css/themes/my-custom-theme.css" />
9     <link rel="stylesheet" href="http://code.jquery.com/mobile/1.2.0/jquery.mobile.structure-1.2.0.min.css" />
10    <script src="http://code.jquery.com/jquery-1.8.2.min.js"></script>
11    <script src="http://code.jquery.com/mobile/1.2.0/jquery.mobile-1.2.0.min.js"></script>
```

```
12
```

```
13 </head>
```



## Final Note

You need to be aware that jQuery Mobile will default to certain swatches when none are specified. For example, page content will default to swatch 'c', list dividers to swatch 'b', etc. As the full jQuery Mobile CSS is replaced by your custom theme CSS and the jQuery Mobile structure CSS, the only swatches available are the ones that you have provided as part of your custom theme. Therefore, you need to either always specify a swatch letter for all your elements or their parent using for example the `data-theme` attribute, or you will need to provide a swatch in your custom theme for the possible defaults. Additionally, the error messages use the swatch 'e', so this should also be specified in your theme.

---

# jQuery API

Traversing > Miscellaneous Traversing

---

## **.add()**

Add elements to the set of matched elements.

Traversing > Miscellaneous Traversing

---

## **.addBack()**

Add the previous set of elements on the stack to the current set, optionally filtered by a selector.

Attributes | Manipulation > Class Attribute | CSS

---

## **.addClass()**

Adds the specified class(es) to each of the set of matched elements.

Manipulation > DOM Insertion, Outside

---

## **.after()**

Insert content, specified by the parameter, after each element in the set of matched elements.

Ajax > Global Ajax Event Handlers

---

## **.ajaxComplete()**

Register a handler to be called when Ajax requests complete. This is an AjaxEvent.

Ajax > Global Ajax Event Handlers

---

## **.ajaxError()**

Register a handler to be called when Ajax requests complete with an error. This is an Ajax Event.

Ajax > Global Ajax Event Handlers

---

## **.ajaxSend()**

Attach a function to be executed before an Ajax request is sent.  
This is an Ajax Event.

Ajax > Global Ajax Event Handlers

---

## **.ajaxStart()**

Register a handler to be called when the first Ajax request begins. This is an Ajax Event.

Ajax > Global Ajax Event Handlers

---

## **.ajaxStop()**

Register a handler to be called when all Ajax requests have completed. This is an Ajax Event.

Ajax > Global Ajax Event Handlers

---

## **.ajaxSuccess()**

Attach a function to be executed whenever an Ajax request completes successfully. This is an Ajax Event.

Selectors > Basic

---

## All Selector (“\*”)

Selects all elements.

[Traversing](#) > [Miscellaneous Traversing](#)

---

## **.andSelf()**

Add the previous set of elements on the stack to the current set.

Effects > Custom

---

## **.animate()**

Perform a custom animation of a set of CSS properties.

[Selectors > Basic Filter](#) | [Selectors > jQuery Extensions](#)

---

## **:animated Selector**

Select all elements that are in the progress of an animation at the time the selector is run.

Manipulation > DOM Insertion, Inside

---

## **.append()**

Insert content, specified by the parameter, to the end of each element in the set of matched elements.

Manipulation > DOM Insertion, Inside

---

## **.appendTo()**

Insert every element in the set of matched elements to the end of the target.

Attributes | Manipulation > General Attributes

---

## **.attr()**

Get the value of an attribute for the first element in the set of matched elements or set one or more attributes for every matched element.

Selectors > Attribute

---

## Attribute Contains Prefix Selector

### `[name="value"]`

Selects elements that have the specified attribute with a value either equal to a given string or starting with that string followed by a hyphen (-).

Selectors > Attribute

---

## Attribute Contains Selector [name\*="value"]

Selects elements that have the specified attribute with a value containing the a given substring.

Selectors > Attribute

---

## Attribute Contains Word Selector

### `[name~="value"]`

Selects elements that have the specified attribute with a value containing a given word, delimited by spaces.

[Selectors](#) > [Attribute](#)

---

## Attribute Ends With Selector

### `[name$="value"]`

Selects elements that have the specified attribute with a value ending exactly with a given string. The comparison is case sensitive.

Selectors > Attribute

---

## Attribute Equals Selector [name="value"]

Selects elements that have the specified attribute with a value exactly equal to a certain value.

[Selectors](#) > [Attribute](#) | [Selectors](#) > [jQuery Extensions](#)

---

## Attribute Not Equal Selector

### [name!="value"]

Select elements that either don't have the specified attribute, or do have the specified attribute but not with a certain value.

[Selectors](#) > [Attribute](#)

---

## Attribute Starts With Selector

### `[name^="value"]`

Selects elements that have the specified attribute with a value beginning exactly with a given string.

[Manipulation](#) > [DOM Insertion, Outside](#)

---

## **.before()**

Insert content, specified by the parameter, before each element in the set of matched elements.

Events > Event Handler Attachment

---

## **.bind()**

Attach a handler to an event for the elements.

Events > Form Events | Forms

---

## **.blur()**

Bind an event handler to the “blur” JavaScript event, or trigger that event on an element.

[Selectors > Form](#) | [Selectors > jQuery Extensions](#)

---

## **:button Selector**

Selects all button elements and elements of type button.

Callbacks Object

---

## callbacks.add()

Add a callback or a collection of callbacks to a callback list.

Callbacks Object

---

## callbacks.disable()

Disable a callback list from doing anything more.

Callbacks Object

---

## callbacks.disabled()

Determine if the callbacks list has been disabled.

Callbacks Object

---

## callbacks.empty()

Remove all of the callbacks from a list.

Callbacks Object

---

## callbacks.fire()

Call all of the callbacks with the given arguments

Callbacks Object

---

## callbacks.fired()

Determine if the callbacks have already been called at least once.

Callbacks Object

---

## callbacks.fireWith()

Call all callbacks in a list with the given context and arguments.

Callbacks Object

---

## callbacks.has()

Determine whether a supplied callback is in a list

Callbacks Object

---

## callbacks.lock()

Lock a callback list in its current state.

Callbacks Object

---

## callbacks.locked()

Determine if the callbacks list has been locked.

Callbacks Object

---

## callbacks.remove()

Remove a callback or a collection of callbacks from a callback list.

Events > Form Events | Forms

---

## **.change()**

Bind an event handler to the “change” JavaScript event, or trigger that event on an element.

[Selectors > Form](#) | [Selectors > jQuery Extensions](#)

---

## **:checkbox Selector**

Selects all elements of type checkbox.

Selectors > Form

---

## **:checked Selector**

Matches all elements that are checked.

Selectors > Hierarchy

---

## `child")" href="0D4EEA27BD.htm" rel=bookmark>Child Selector (“parent > child”)`

Selects all direct child elements specified by “child” of elements specified by “parent”.

Traversing > Tree Traversal

---

## **.children()**

Get the children of each element in the set of matched elements, optionally filtered by a selector.

Selectors > Basic

---

## Class Selector (“`.class`”)

Selects all elements with the given class.

Effects > Custom | Data | Utilities

---

## `.clearQueue()`

Remove from the queue all items that have not yet been run.

Events > Mouse Events

---

## `.click()`

Bind an event handler to the “click” JavaScript event, or trigger that event on an element.

Manipulation > Copying

---

## `.clone()`

Create a deep copy of the set of matched elements.

[Traversing](#) > [Tree Traversal](#)

---

## `.closest()`

For each element in the set, get the first element that matches the selector by testing the element itself and traversing up through its ancestors in the DOM tree.

Selectors > Content Filter

---

## **:contains() Selector**

Select all elements that contain the specified text.

Traversing > Miscellaneous Traversing

---

## **.contents()**

Get the children of each element in the set of matched elements, including text and comment nodes.

[Internals](#) | [Properties](#) > [Properties of jQuery Object Instances](#)

---

## **.context**

The DOM node context originally passed to `jQuery()`; if none was passed then context will likely be the document.

CSS | Manipulation > Style Properties

---

## `.css()`

Get the value of a style property for the first element in the set of matched elements or set one or more CSS properties for every matched element.

[Data](#) | [Miscellaneous](#) > [Data Storage](#)

---

## **.data()**

Store arbitrary data associated with the matched elements or return the value at the named data store for the first element in the set of matched elements.

Events > Mouse Events

---

## **.dblclick()**

Bind an event handler to the “dblclick” JavaScript event, or trigger that event on an element.

Deferred Object

---

## **deferred.always()**

Add handlers to be called when the Deferred object is either resolved or rejected.

Deferred Object

---

## deferred.done()

Add handlers to be called when the Deferred object is resolved.

Deferred Object

---

## deferred.fail()

Add handlers to be called when the Deferred object is rejected.

Deferred Object

---

## deferred.isRejected()

Determine whether a Deferred object has been rejected.

Deferred Object

---

## deferred.isResolved()

Determine whether a Deferred object has been resolved.

Deferred Object

---

## deferred.notify()

Call the progressCallbacks on a Deferred object with the given args.

Deferred Object

---

## deferred.notifyWith()

Call the progressCallbacks on a Deferred object with the given context and args.

Deferred Object

---

## deferred.pipe()

Utility method to filter and/or chain Deferreds.

Deferred Object

---

## deferred.progress()

Add handlers to be called when the Deferred object generates progress notifications.

Deferred Object

---

## deferred.promise()

Return a Deferred's Promise object.

Deferred Object

---

## deferred.reject()

Reject a Deferred object and call any failCallbacks with the given args.

Deferred Object

---

## deferred.rejectWith()

Reject a Deferred object and call any failCallbacks with the given context and args.

Deferred Object

---

## deferred.resolve()

Resolve a Deferred object and call any doneCallbacks with the given args.

Deferred Object

---

## deferred.resolveWith()

Resolve a Deferred object and call any doneCallbacks with the given context and args.

Deferred Object

---

## **deferred.state()**

Determine the current state of a Deferred object.

Deferred Object

---

## deferred.then()

Add handlers to be called when the Deferred object is resolved, rejected, or still in progress.

Effects > Custom

---

## **.delay()**

Set a timer to delay execution of subsequent items in the queue.

Events > Event Handler Attachment

---

## **.delegate()**

Attach a handler to one or more events for all elements that match the selector, now or in the future, based on a specific set of root elements.

Effects > Custom | Data | Utilities

---

## **.dequeue()**

Execute the next function on the queue for the matched elements.

Selectors > Hierarchy

---

## Descendant Selector (“ancestor descendant”)

Selects all elements that are descendants of a given ancestor.

Manipulation > DOM Removal

---

---

## **.detach()**

Remove the set of matched elements from the DOM.

Events > Event Handler Attachment

---

## **.die()**

Remove event handlers previously attached using `.live()` from the elements.

Selectors > Form

---

## **:disabled Selector**

Selects all elements that are disabled.

Miscellaneous > Collection Manipulation | Traversing

---

## `.each()`

Iterate over a jQuery object, executing a function for each matched element.

Selectors > Basic

---

## Element Selector (“element”)

Selects all elements with the given tag name.

Manipulation > DOM Removal

---

## **.empty()**

Remove all child nodes of the set of matched elements from the DOM.

Selectors > Content Filter

---

## **:empty Selector**

Select all elements that have no children (including text nodes).

Selectors > Form

---

## **:enabled Selector**

Selects all elements that are enabled.

[Traversing](#) > [Miscellaneous Traversing](#)

---

## **.end()**

End the most recent filtering operation in the current chain and return the set of matched elements to its previous state.

Traversing > Filtering

---

## **.eq()**

Reduce the set of matched elements to the one at the specified index.

[Selectors > Basic Filter](#) | [Selectors > jQuery Extensions](#)

---

## **:eq() Selector**

Select the element at index n within the matched set.

Events > Browser Events

---

## **.error()**

Bind an event handler to the “error” JavaScript event.

[Selectors](#) > [Basic Filter](#) | [Selectors](#) > [jQuery Extensions](#)

---

## **:even Selector**

Selects even elements, zero-indexed. See also [odd](#).

[Events](#) > [Event Object](#)

---

## **event.currentTarget**

The current DOM element within the event bubbling phase.

Events > Event Object

---

## event.data

An optional object of data passed to an event method when the current executing handler is bound.

Events > Event Object | Events

---

## event.delegateTarget

The element where the currently-called jQuery event handler was attached.

Events > Event Object

---

## **event.isDefaultPrevented()**

Returns whether `event.preventDefault()` was ever called on this event object.

Events > Event Object

---

## **event.isImmediatePropagationStopped()**

Returns whether `event.stopImmediatePropagation()` was ever called on this event object.

Events > Event Object

---

## **event.isPropagationStopped()**

Returns whether `event.stopPropagation()` was ever called on this event object.

Events > Event Object

---

## event.metaKey

Indicates whether the META key was pressed when the event fired.

Events > Event Object

---

## **event.namespace**

The namespace specified when the event was triggered.

Events > Event Object

---

## **event.pageX**

The mouse position relative to the left edge of the document.

Events > Event Object

---

## event.pageY

The mouse position relative to the top edge of the document.

Events > Event Object

---

## **event.preventDefault()**

If this method is called, the default action of the event will not be triggered.

Events > Event Object

---

## **event.relatedTarget**

The other DOM element involved in the event, if any.

Events > Event Object

---

## **event.result**

The last value returned by an event handler that was triggered by this event, unless the value was undefined.

Events > Event Object

---

## **event.stopImmediatePropagation()**

Keeps the rest of the handlers from being executed and prevents the event from bubbling up the DOM tree.

Events > Event Object

---

## **event.stopPropagation()**

Prevents the event from bubbling up the DOM tree, preventing any parent handlers from being notified of the event.

Events > Event Object

---

## event.target

The DOM element that initiated the event.

Events > Event Object

---

## event.timeStamp

The difference in milliseconds between the time the browser created the event and January 1, 1970.

Events > Event Object

---

## event.type

Describes the nature of the event.

Events > Event Object

---

## event.which

For key or mouse events, this property indicates the specific key or button that was pressed.

Effects > Fading

---

## **.fadeIn()**

Display the matched elements by fading them to opaque.

Effects > Fading

---

## **.fadeOut()**

Hide the matched elements by fading them to transparent.

Effects > Fading

---

## `.fadeTo()`

Adjust the opacity of the matched elements.

Effects | Effects > Fading

---

## **.fadeToggle()**

Display or hide the matched elements by animating their opacity.

[Selectors > Form](#) | [Selectors > jQuery Extensions](#)

---

## **:file Selector**

Selects all elements of type file.

Traversing > Filtering

---

## **.filter()**

Reduce the set of matched elements to those that match the selector or pass the function's test.

Traversing > Tree Traversal

---

## **.find()**

Get the descendants of each element in the current set of matched elements, filtered by a selector, jQuery object, or element.

Effects > Custom

---

## **.finish()**

Stop the currently-running animation, remove all queued animations, and complete all animations for the matched elements.

Traversing > Filtering

---

## **.first()**

Reduce the set of matched elements to the first in the set.

Selectors > Child Filter

---

## **:first-child Selector**

Selects all elements that are the first child of their parent.

Selectors > Child Filter

---

## **:first-of-type Selector**

Selects all elements that are the first among siblings of the same element name.

[Selectors > Basic Filter](#) | [Selectors > jQuery Extensions](#)

---

## **:first Selector**

Selects the first matched element.

Events > Form Events | Forms

---

## **.focus()**

Bind an event handler to the “focus” JavaScript event, or trigger that event on an element.

[Selectors > Basic Filter](#) | [Selectors > Form](#)

---

## **:focus Selector**

Selects element if it is currently focused.

[Events > Keyboard Events](#) | [Events > Mouse Events](#)

---

## **.focusin()**

Bind an event handler to the “focusin” event.

[Events > Keyboard Events](#) | [Events > Mouse Events](#)

---

## **.focusout()**

Bind an event handler to the “focusout” JavaScript event.

Miscellaneous > DOM Element Methods

---

## **.get()**

Retrieve the DOM elements matched by the jQuery object.

[Selectors > Basic Filter](#) | [Selectors > jQuery Extensions](#)

---

## **:gt() Selector**

Select all elements at an index greater than index within the matched set.

Traversing > Filtering

---

## `.has()`

Reduce the set of matched elements to those that have a descendant that matches the selector or DOM element.

Selectors > Attribute

---

## Has Attribute Selector [name]

Selects elements that have the specified attribute, with any value.

[Selectors](#) > [Content Filter](#) | [Selectors](#) > [jQuery Extensions](#)

---

## **:has() Selector**

Selects elements which contain at least one element that matches the specified selector.

[Attributes](#) | [Manipulation > Class Attribute](#) | [CSS](#)

---

## **.hasClass()**

Determine whether any of the matched elements are assigned the given class.

[Selectors > Basic Filter](#) | [Selectors > jQuery Extensions](#)

---

## **:header Selector**

Selects all elements that are headers, like h1, h2, h3 and so on.

[CSS](#) | [Dimensions](#) | [Manipulation](#) > [Style Properties](#)

---

## **.height()**

Get the current computed height for the first element in the set of matched elements or set the height of every matched element.

[Selectors > jQuery Extensions](#) | [Selectors > Visibility Filter](#)

---

## **:hidden Selector**

Selects all elements that are hidden.

Effects > Basics

---

## **.hide()**

Hide the matched elements.

Events > Mouse Events

---

## **.hover()**

Bind one or two handlers to the matched elements, to be executed when the mouse pointer enters and leaves the elements.

Attributes | Manipulation > DOM Insertion, Inside

---

## `.html()`

Get the HTML contents of the first element in the set of matched elements or set the HTML contents of every matched element.

Selectors > Basic

---

## ID Selector (“#id”)

Selects a single element with the given id attribute.

[Selectors > Form](#) | [Selectors > jQuery Extensions](#)

---

## **:image Selector**

Selects all elements of type image.

Miscellaneous > DOM Element Methods

---

## `.index()`

Search for a given element from among the matched elements.

[CSS](#) | [Dimensions](#) | [Manipulation](#) > [Style Properties](#)

---

## **.innerHeight()**

Get the current computed height for the first element in the set of matched elements, including padding but not border.

CSS | Dimensions | Manipulation > Style Properties

---

## **.innerWidth()**

Get the current computed width for the first element in the set of matched elements, including padding but not border.

[Selectors > Form](#) | [Selectors > jQuery Extensions](#)

---

## **:input Selector**

Selects all input, textarea, select and button elements.

Manipulation > DOM Insertion, Outside

---

## **.insertAfter()**

Insert every element in the set of matched elements after the target.

Manipulation > DOM Insertion, Outside

---

## **.insertBefore()**

Insert every element in the set of matched elements before the target.

Traversing > Filtering

---

## **.is()**

Check the current matched set of elements against a selector, element, or jQuery object and return true if at least one of these elements matches the given arguments.

Core

---

## jQuery()

Return a collection of matched elements either found in the DOM based on passed argument(s) or created by passing an HTML string.

[Internals](#) | [Properties](#) > [Properties of jQuery Object Instances](#)

---

## **.jquery**

A string containing the jQuery version number.

Ajax > Low-Level Interface

---

## jQuery.ajax()

Perform an asynchronous HTTP (Ajax) request.

[Ajax](#) > [Low-Level Interface](#)

---

## jQuery.ajaxPrefilter()

Handle custom Ajax options or modify existing options before each request is sent and before they are processed by \$.ajax().

Ajax > Low-Level Interface

---

## jQuery.ajaxSetup()

Set default values for future Ajax requests.

[Ajax](#) > [Low-Level Interface](#)

---

## jQuery.ajaxTransport()

Creates an object that handles the actual transmission of Ajax data.

Utilities

---

## jQuery.boxModel

Deprecated in jQuery 1.3 (see [jQuery.support](#)). States if the current page, in the user's browser, is being rendered using the W3C CSS Box Model.

[Properties](#) > [Properties of the Global jQuery Object](#) | [Utilities](#)

---

## jQuery.browser

Contains flags for the useragent, read from navigator.userAgent. We recommend against using this property; please try to use feature detection instead (see jQuery.support). jQuery.browser may be moved to a plugin in a future release of jQuery.

Callbacks Object

---

## jQuery.Callbacks()

A multi-purpose callbacks list object that provides a powerful way to manage callback lists.

Utilities

---

## jQuery.contains()

Check to see if a DOM element is a descendant of another DOM element.

CSS

---

## jQuery.cssHooks

Hook directly into jQuery to override how particular CSS properties are retrieved or set, normalize CSS property naming, or create custom properties.

Data | Utilities

---

## jQuery.data()

Store arbitrary data associated with the specified element and/or return the value that was set.

Deferred Object

---

## jQuery.Deferred()

A constructor function that returns a chainable utility object with methods to register multiple callbacks into callback queues, invoke callback queues, and relay the success or failure state of any synchronous or asynchronous function.

Data | Utilities

---

## jQuery.dequeue()

Execute the next function on the queue for the matched element.

Utilities

---

## jQuery.each()

A generic iterator function, which can be used to seamlessly iterate over both objects and arrays. Arrays and array-like objects with a length property (such as a function's arguments object) are iterated by numeric index, from 0 to length-1. Other objects are iterated via their named properties.

Internals

---

## jQuery.error()

Takes a string and throws an exception containing it.

Utilities

---

## jQuery.extend()

Merge the contents of two or more objects together into the first object.

Effects > Custom | Properties > Properties of the Global jQuery Object

---

## jQuery.fx.interval

The rate (in milliseconds) at which animations fire.

[Effects](#) > [Custom](#) | [Properties](#) > [Properties of the Global jQuery Object](#)

---

# jQuery.fx.off

Globally disable all animations.

[Ajax](#) > [Shorthand Methods](#)

---

## jQuery.get()

Load data from the server using a HTTP GET request.

[Ajax](#) > [Shorthand Methods](#)

---

## jQuery.getJSON()

Load JSON-encoded data from the server using a GET HTTP request.

[Ajax](#) > [Shorthand Methods](#)

---

## jQuery.getScript()

Load a JavaScript file from the server using a GET HTTP request, then execute it.

Utilities

---

## jQuery.globalEval()

Execute some JavaScript code globally.

Utilities

---

## jQuery.grep()

Finds the elements of an array which satisfy a filter function. The original array is not affected.

Data

---

## jQuery.hasData()

Determine whether an element has any jQuery data associated with it.

Core

---

## jQuery.holdReady()

Holds or releases the execution of jQuery's ready event.

Utilities

---

## jQuery.inArray()

Search for a specified value within an array and return its index (or -1 if not found).

Utilities

---

## jQuery.isArray()

Determine whether the argument is an array.

Utilities

---

## jQuery.isEmptyObject()

Check to see if an object is empty (contains no enumerable properties).

Utilities

---

## jQuery.isFunction()

Determine if the argument passed is a Javascript function object.

Utilities

---

## jQuery.isNumeric()

Determines whether its argument is a number.

Utilities

---

## jQuery.isPlainObject()

Check to see if an object is a plain object (created using “{}” or “new Object”).

Utilities

---

## jQuery.isWindow()

Determine whether the argument is a window.

Utilities

---

## jQuery.isXMLDoc()

Check to see if a DOM node is within an XML document (or is an XML document).

Utilities

---

## jQuery.makeArray()

Convert an array-like object into a true JavaScript array.

Utilities

---

## jQuery.map()

Translate all items in an array or object to new array of items.

Utilities

---

## jQuery.merge()

Merge the contents of two arrays together into the first array.

[Core](#) | [Miscellaneous](#) > [Setup Methods](#)

---

## jQuery.noConflict()

Relinquish jQuery's control of the \$ variable.

Utilities

---

# jQuery.noop()

An empty function.

Utilities

---

## jQuery.now()

Return a number representing the current time.

[Miscellaneous](#) > [Collection Manipulation](#) | [Forms](#) | [Ajax](#) > [Helper Functions](#)

---

## jQuery.param()

Create a serialized representation of an array or object, suitable for use in a URL query string or Ajax request.

Utilities

---

## jQuery.parseHTML()

Parses a string into an array of DOM nodes.

Utilities

---

## jQuery.parseJSON()

Takes a well-formed JSON string and returns the resulting JavaScript object.

Utilities

---

## jQuery.parseXML()

Parses a string into an XML document.

[Ajax](#) > [Shorthand Methods](#)

---

## jQuery.post()

Load data from the server using a HTTP POST request.

[Events](#) > [Event Handler Attachment](#) | [Utilities](#)

---

## jQuery.proxy()

Takes a function and returns a new one that will always have a particular context.

Data | Utilities

---

## jQuery.queue()

Show or manipulate the queue of functions to be executed on the matched element.

Data | Utilities

---

## jQuery.removeData()

Remove a previously-stored piece of data.

Core

---

## jQuery.sub()

Creates a new copy of jQuery whose properties and methods can be modified without affecting the original jQuery object.

[Properties](#) > [Properties of the Global jQuery Object](#) | [Utilities](#)

---

## jQuery.support

A collection of properties that represent the presence of different browser features or bugs. Primarily intended for jQuery's internal use; specific properties may be removed when they are no longer needed internally to improve page startup performance.

Utilities

---

## jQuery.trim()

Remove the whitespace from the beginning and end of a string.

Utilities

---

## jQuery.type()

Determine the internal JavaScript [[Class]] of an object.

Utilities

---

## jQuery.unique()

Sorts an array of DOM elements, in place, with the duplicates removed. Note that this only works on arrays of DOM elements, not strings or numbers.

Core | Deferred Object

---

## jQuery.when()

Provides a way to execute callback functions based on one or more objects, usually Deferred objects that represent asynchronous events.

Events > Keyboard Events

---

## **.keydown()**

Bind an event handler to the “keydown” JavaScript event, or trigger that event on an element.

Events > Keyboard Events

---

## **.keypress()**

Bind an event handler to the “keypress” JavaScript event, or trigger that event on an element.

Events > Keyboard Events

---

## **.keyup()**

Bind an event handler to the “keyup” JavaScript event, or trigger that event on an element.

Selectors > Basic Filter

---

## **:lang Selector**

Selects all elements of the specified language.

Traversing > Filtering

---

## **.last()**

Reduce the set of matched elements to the final one in the set.

Selectors > Child Filter

---

## **:last-child Selector**

Selects all elements that are the last child of their parent.

Selectors > Child Filter

---

## **:last-of-type Selector**

Selects all elements that are the last among siblings of the same element name.

[Selectors > Basic Filter](#) | [Selectors > jQuery Extensions](#)

---

## **:last Selector**

Selects the last matched element.

[Properties](#) > [Properties of jQuery Object Instances](#)

---

## **.length**

The number of elements in the jQuery object.

Events > Event Handler Attachment

---

## **.live()**

Attach an event handler for all elements which match the current selector, now and in the future.

[Ajax](#) > [Shorthand Methods](#)

---

## **.load()**

Load data from the server and place the returned HTML into the matched element.

Events > Document Loading

---

## **.load()**

Bind an event handler to the “load” JavaScript event.

[Selectors > Basic Filter](#) | [Selectors > jQuery Extensions](#)

---

## **:lt() Selector**

Select all elements at an index less than index within the matched set.

Traversing > Filtering

---

## **.map()**

Pass each element in the current matched set through a function, producing a new jQuery object containing the return values.

Events > Mouse Events

---

## **.mousedown()**

Bind an event handler to the “mousedown” JavaScript event, or trigger that event on an element.

Events > Mouse Events

---

## **.mouseenter()**

Bind an event handler to be fired when the mouse enters an element, or trigger that handler on an element.

Events > Mouse Events

---

## **.mouseleave()**

Bind an event handler to be fired when the mouse leaves an element, or trigger that handler on an element.

Events > Mouse Events

---

## `.mousemove()`

Bind an event handler to the “mousemove” JavaScript event, or trigger that event on an element.

Events > Mouse Events

---

## **.mouseout()**

Bind an event handler to the “mouseout” JavaScript event, or trigger that event on an element.

Events > Mouse Events

---

## **.mouseover()**

Bind an event handler to the “mouseover” JavaScript event, or trigger that event on an element.

Events > Mouse Events

---

## **.mouseup()**

Bind an event handler to the “mouseup” JavaScript event, or trigger that event on an element.

Selectors > Attribute

---

## Multiple Attribute Selector [name="value"] [name2="value2"]

Matches elements that match all of the specified attribute filters.

Selectors > Basic

---

---

## Multiple Selector (“selector1, selector2, selectorN”)

Selects the combined results of all the specified selectors.

[Traversing](#) > [Tree Traversal](#)

---

---

## `.next()`

Get the immediately following sibling of each element in the set of matched elements. If a selector is provided, it retrieves the next sibling only if it matches that selector.

Selectors > Hierarchy

---

## Next Adjacent Selector (“prev + next”)

Selects all next elements matching “next” that are immediately preceded by a sibling “prev”.

Selectors > Hierarchy

---

## Next Siblings Selector (“prev ~ siblings”)

Selects all sibling elements that follow after the “prev” element, have the same parent, and match the filtering “siblings” selector.

Traversing > Tree Traversal

---

## **.nextAll()**

Get all following siblings of each element in the set of matched elements, optionally filtered by a selector.

Traversing > Tree Traversal

---

## **.nextUntil()**

Get all following siblings of each element up to but not including the element matched by the selector, DOM node, or jQuery object passed.

[Traversing > Filtering](#) | [Traversing > Miscellaneous Traversing](#)

---

## **.not()**

Remove elements from the set of matched elements.

Selectors > Basic Filter

---

## **:not() Selector**

Selects all elements that do not match the given selector.

Selectors > Child Filter

---

## **:nth-child() Selector**

Selects all elements that are the nth-child of their parent.

Selectors > Child Filter

---

## **:nth-last-child()** Selector

Selects all elements that are the nth-child of their parent, counting from the last element to the first.

Selectors > Child Filter

---

## **:nth-last-of-type()** Selector

Selects all elements that are the nth-child of their parent, counting from the last element to the first.

Selectors > Child Filter

---

## **:nth-of-type() Selector**

Selects all elements that are the nth child of their parent in relation to siblings with the same element name.

[Selectors > Basic Filter](#) | [Selectors > jQuery Extensions](#)

---

## **:odd Selector**

Selects odd elements, zero-indexed. See also even.

Events > Event Handler Attachment

---

## `.off()`

Remove an event handler.

[CSS](#) | [Offset](#) | [Manipulation](#) > [Style Properties](#)

---

## **.offset()**

Get the current coordinates of the first element, or set the coordinates of every element, in the set of matched elements, relative to the document.

[Offset](#) | [Traversing](#) > [Tree Traversal](#)

---

## **.offsetParent()**

Get the closest ancestor element that is positioned.

Events > Event Handler Attachment

---

## `.on()`

Attach an event handler function for one or more events to the selected elements.

Events > Event Handler Attachment

---

## **.one()**

Attach a handler to an event for the elements. The handler is executed at most once per element.

Selectors > Child Filter

---

## **:only-child Selector**

Selects all elements that are the only child of their parent.

Selectors > Child Filter

---

## **:only-of-type Selector**

Selects all elements that have no siblings with the same element name.

CSS | Dimensions | Manipulation > Style Properties

---

## **.outerHeight()**

Get the current computed height for the first element in the set of matched elements, including padding, border, and optionally margin. Returns an integer (without "px") representation of the value or null if called on an empty set of elements.

[CSS](#) | [Dimensions](#) | [Manipulation](#) > [Style Properties](#)

---

## **.outerWidth()**

Get the current computed width for the first element in the set of matched elements, including padding and border.

Traversing > Tree Traversal

---

## **.parent()**

Get the parent of each element in the current set of matched elements, optionally filtered by a selector.

[Selectors > Content Filter](#) | [Selectors > jQuery Extensions](#)

---

## **:parent Selector**

Select all elements that that have at least one child node (either an element or text).

Traversing > Tree Traversal

---

## **.parents()**

Get the ancestors of each element in the current set of matched elements, optionally filtered by a selector.

Traversing > Tree Traversal

---

## **.parentsUntil()**

Get the ancestors of each element in the current set of matched elements, up to but not including the element matched by the selector, DOM node, or jQuery object.

[Selectors > Form](#) | [Selectors > jQuery Extensions](#)

---

## **:password Selector**

Selects all elements of type password.

[CSS](#) | [Offset](#) | [Manipulation](#) > [Style Properties](#)

---

## **.position()**

Get the current coordinates of the first element in the set of matched elements, relative to the offset parent.

Manipulation > DOM Insertion, Inside

---

## **.prepend()**

Insert content, specified by the parameter, to the beginning of each element in the set of matched elements.

Manipulation > DOM Insertion, Inside

---

## **.prependTo()**

Insert every element in the set of matched elements to the beginning of the target.

Traversing > Tree Traversal

---

## `.prev()`

Get the immediately preceding sibling of each element in the set of matched elements, optionally filtered by a selector.

Traversing > Tree Traversal

---

## **.prevAll()**

Get all preceding siblings of each element in the set of matched elements, optionally filtered by a selector.

Traversing > Tree Traversal

---

## **.prevUntil()**

Get all preceding siblings of each element up to but not including the element matched by the selector, DOM node, or jQuery object.

Deferred Object

---

## **.promise()**

Return a Promise object to observe when all actions of a certain type bound to the collection, queued or not, have finished.

Attributes | Manipulation > General Attributes

---

## **.prop()**

Get the value of a property for the first element in the set of matched elements or set one or more properties for every matched element.

Internals

---

## **.pushStack()**

Add a collection of DOM elements onto the jQuery stack.

[Effects](#) > [Custom](#) | [Data](#) | [Utilities](#)

---

## `.queue()`

Show or manipulate the queue of functions to be executed on the matched elements.

[Selectors > Form](#) | [Selectors > jQuery Extensions](#)

---

## **:radio Selector**

Selects all elements of type radio.

Events > Document Loading

---

## **.ready()**

Specify a function to execute when the DOM is fully loaded.

Manipulation > DOM Removal

---

## `.remove()`

Remove the set of matched elements from the DOM.

[Attributes](#) | [Manipulation](#) > [General Attributes](#)

---

## `.removeAttr()`

Remove an attribute from each element in the set of matched elements.

Attributes | Manipulation > Class Attribute | CSS

---

## **.removeClass()**

Remove a single class, multiple classes, or all classes from each element in the set of matched elements.

Data | Miscellaneous > Data Storage

---

## **.removeData()**

Remove a previously-stored piece of data.

Attributes | Manipulation > General Attributes

---

## **.removeProp()**

Remove a property for the set of matched elements.

Manipulation > DOM Replacement

---

## `.replaceAll()`

Replace each target element with the set of matched elements.

Manipulation > DOM Replacement

---

## **.replaceWith()**

Replace each element in the set of matched elements with the provided new content and return the set of elements that was removed.

[Selectors > Form](#) | [Selectors > jQuery Extensions](#)

---

## **:reset Selector**

Selects all elements of type reset.

Events > Browser Events

---

## **.resize()**

Bind an event handler to the “resize” JavaScript event, or trigger that event on an element.

[Selectors](#) > [Basic Filter](#)

---

## **:root Selector**

Selects the element that is the root of the document.

Events > Browser Events

---

## **.scroll()**

Bind an event handler to the “scroll” JavaScript event, or trigger that event on an element.

CSS | Offset | Manipulation > Style Properties

---

## `.scrollLeft()`

Get the current horizontal position of the scroll bar for the first element in the set of matched elements or set the horizontal position of the scroll bar for every matched element.

CSS | Offset | Manipulation > Style Properties

---

## `.scrollTop()`

Get the current vertical position of the scroll bar for the first element in the set of matched elements or set the vertical position of the scroll bar for every matched element.

[Events](#) > [Form Events](#) | [Forms](#)

---

## **.select()**

Bind an event handler to the “select” JavaScript event, or trigger that event on an element.

[Selectors > Form](#) | [Selectors > jQuery Extensions](#)

---

## **:selected Selector**

Selects all elements that are selected.

[Internals](#) | [Properties](#) > [Properties of the Global jQuery Object](#)

---

## **.selector**

A selector representing selector originally passed to jQuery().

Forms | Ajax > Helper Functions

---

## **.serialize()**

Encode a set of form elements as a string for submission.

Forms | [Ajax](#) > [Helper Functions](#)

---

## **.serializeArray()**

Encode a set of form elements as an array of names and values.

Effects > Basics

---

## `.show()`

Display the matched elements.

[Traversing](#) > [Tree Traversal](#)

---

## **.siblings()**

Get the siblings of each element in the set of matched elements, optionally filtered by a selector.

Miscellaneous > DOM Element Methods

---

## **.size()**

Return the number of elements in the jQuery object.

Traversing > Filtering

---

## **.slice()**

Reduce the set of matched elements to a subset specified by a range of indices.

Effects > Sliding

---

## **.slideDown()**

Display the matched elements with a sliding motion.

Effects > Sliding

---

## **.slideToggle()**

Display or hide the matched elements with a sliding motion.

Effects > Sliding

---

## `.slideUp()`

Hide the matched elements with a sliding motion.

Effects > Custom

---

## **.stop()**

Stop the currently-running animation on the matched elements.

Events > Form Events | Forms

---

## **.submit()**

Bind an event handler to the “submit” JavaScript event, or trigger that event on an element.

[Selectors > Form](#) | [Selectors > jQuery Extensions](#)

---

## **:submit Selector**

Selects all elements of type submit.

Selectors > Basic Filter

---

## **:target Selector**

Selects the target element indicated by the fragment identifier of the document's URI.

Manipulation > DOM Insertion, Inside

---

## **.text()**

Get the combined text contents of each element in the set of matched elements, including their descendants, or set the text contents of the matched elements.

[Selectors > Form](#) | [Selectors > jQuery Extensions](#)

---

## **:text Selector**

Selects all elements of type text.

Miscellaneous > DOM Element Methods

---

## **.toArray()**

Retrieve all the DOM elements contained in the jQuery set, as an array.

Effects > Basics

---

## **.toggle()**

Display or hide the matched elements.

[Deprecated](#) > [Deprecated 1.8](#) | [Events](#) > [Mouse Events](#)

---

## **.toggle()**

Bind two or more handlers to the matched elements, to be executed on alternate clicks.

Attributes | Manipulation > Class Attribute | CSS

---

## **.toggleClass()**

Add or remove one or more classes from each element in the set of matched elements, depending on either the class's presence or the value of the switch argument.

Events > Event Handler Attachment

---

## **.trigger()**

Execute all handlers and behaviors attached to the matched elements for the given event type.

Events > Event Handler Attachment

---

## **.triggerHandler()**

Execute all handlers attached to an element for an event.

Events > Event Handler Attachment

---

## **.unbind()**

Remove a previously-attached event handler from the elements.

Events > Event Handler Attachment

---

## **.undelegate()**

Remove a handler from the event for all elements which match the current selector, based upon a specific set of root elements.

Events > Document Loading

---

## **.unload()**

Bind an event handler to the “unload” JavaScript event.

[Manipulation > DOM Insertion, Around](#) | [Manipulation > DOM Removal](#)

---

## `.unwrap()`

Remove the parents of the set of matched elements from the DOM, leaving the matched elements in their place.

[Attributes](#) | [Forms](#) | [Manipulation](#) > [General Attributes](#)

---

## **.val()**

Get the current value of the first element in the set of matched elements or set the value of every matched element.

[Selectors > jQuery Extensions](#) | [Selectors > Visibility Filter](#)

---

## **:visible Selector**

Selects all elements that are visible.

[CSS](#) | [Dimensions](#) | [Manipulation](#) > [Style Properties](#)

---

## `.width()`

Get the current computed width for the first element in the set of matched elements or set the width of every matched element.

Manipulation > DOM Insertion, Around

---

## `.wrap()`

Wrap an HTML structure around each element in the set of matched elements.

Manipulation > DOM Insertion, Around

---

## **.wrapAll()**

Wrap an HTML structure around all elements in the set of matched elements.

Manipulation > DOM Insertion, Around

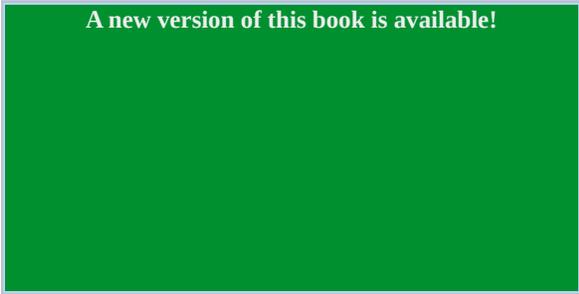
---

## `.wrapInner()`

Wrap an HTML structure around the content of each element in the set of matched elements.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

# .add()

Categories: [Traversing](#) > [Miscellaneous Traversing](#)

`.add( selector )`

Returns: [jQuery](#)

**Description:** *Add elements to the set of matched elements.*

**.add( selector )**

version added: [1.0](#)

**selector**

Type: [Selector](#)

A string representing a selector expression to find additional elements to add to the set of matched elements.

**.add( elements )**

version added: [1.0](#)

**elements**

Type: [Elements](#)

One or more elements to add to the set of matched elements.

**.add( html )**

version added: [1.0](#)

**html**

Type: [HTML](#)

An HTML fragment to add to the set of matched elements.

**.add( jQuery object )**

version added: [1.3.2](#)

**jQuery object**

Type: [jQuery object](#)

An existing jQuery object to add to the set of matched elements.

**.add( selector, context )**

version added: [1.4](#)

**selector**

Type: [Selector](#)

A string representing a selector expression to find additional elements to add to the set of matched elements.

**context**

Type: [Element](#)

The point in the document at which the selector should begin

matching; similar to the context argument of the `$(selector, context)` method.

Given a jQuery object that represents a set of DOM elements, the `.add()` method constructs a new jQuery object from the union of those elements and the ones passed into the method. The argument to `.add()` can be pretty much anything that `$( )` accepts, including a jQuery selector expression, references to DOM elements, or an HTML snippet.

The updated set of elements can be used in a following (chained) method, or assigned to a variable for later use. For example:

```
1 | $("p").add("div").addClass("widget");
2 | var pdiv = $("p").add("div");
```

The following will *not* save the added elements, because the `.add()` method creates a new set and leaves the original set in `pdiv` unchanged:

```
1 | var pdiv = $("p");
2 | pdiv.add("div"); // WRONG, pdiv will not change
```

Consider a page with a simple list and a paragraph following it:

```
1 | <ul>
2 |   <li>list item 1</li>
3 |   <li>list item 2</li>
4 |   <li>list item 3</li>
5 | </ul>
6 | <p>a paragraph</p>
```

We can select the list items and then the paragraph by using either a selector or a reference to the DOM element itself as the `.add()`

method's argument:

```
1 | $('li').add('p').css('background-color', 'red')
```

Or:

```
1 | $('li').add(document.getElementsByTagName('p'))  
2 |   .css('background-color', 'red');
```

The result of this call is a red background behind all four elements. Using an HTML snippet as the `.add()` method's argument (as in the third version), we can create additional elements on the fly and add those elements to the matched set of elements. Let's say, for example, that we want to alter the background of the list items along with a newly created paragraph:

```
1 | $('li').add('<p id="new">new paragraph</p>')  
2 |   .css('background-color', 'red');
```

Although the new paragraph has been created and its background color changed, it still does not appear on the page. To place it on the page, we could add one of the insertion methods to the chain.

As of jQuery 1.4 the results from `.add()` will always be returned in document order (rather than a simple concatenation).

**Note:** To reverse the `.add()` you can use `.not( elements | selector )` to remove elements from the jQuery results, or `.end()` to return to the selection before you added.

## Examples:

**Example:** Finds all divs and makes a border. Then adds all paragraphs to the jQuery object to set their backgrounds yellow.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     div { width:60px; height:60px; margin:10px;
6     p { clear:left; font-weight:bold; font-size:
7       color:blue; margin:0 10px; padding:2px;
8   </style>
9   <script src="http://code.jquery.com/jquery-
10 </head>
11 <body>
12   <div></div>
13
14   <div></div>
15   <div></div>
16   <div></div>
17   <div></div>
18   <div></div>
19
20   <p>Added this... (notice no border)</p>
21 <script>
22
23   $("div").css("border", "2px solid red")
24     .add("p")
25     .css("background", "yellow");
26 </script>
27
28 </body>
29 </html>
```

## Demo

**Example:** Adds more elements, matched by the given expression, to the set of matched elements.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
5 </head>
6 <body>
7   <p>Hello</p><span>Hello Again</span>
8   <script>$("p").add("span").css("background-color", "red");</script>
9
10 </body>
11 </html>
```

## Demo

**Example:** Adds more elements, created on the fly, to the set of matched elements.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
5 </head>
6 <body>
7   <p>Hello</p>
8   <script>$("p").clone().add("<span>Again</span>");</script>
9
10 </body>
11 </html>
```

## Demo

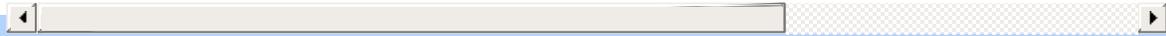
**Example** Adds one or more Elements to the set of matched elements.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
5 </head>
6 <body>
7   <p>Hello</p><span id="a">Hello Again</span>
8   <script>$("p").add(document.getElementById("a"))
9
10  </body>
11 </html>
```

## Demo

**Example** Demonstrates how to add (or push) elements to an existing collection

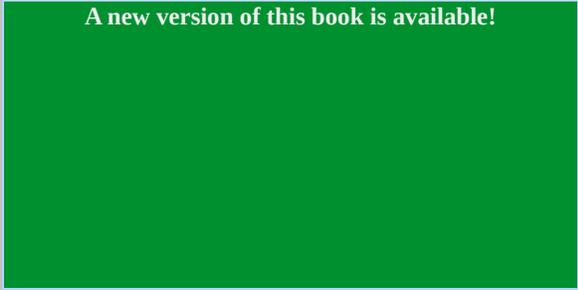
```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
5 </head>
6 <body>
7   <p>Hello</p><span id="a">Hello Again</span>
8   <script>var collection = $("p");
9   // capture the new collection
10  collection = collection.add(document.getElementById("a"));
11  collection.css("background", "yellow");</script>
12
13 </body>
14 </html>
```



Demo

POWERED BY **HERONOTE**

A new version of this book is available!



---

# .addBack()

Categories: [Traversing](#) > [Miscellaneous Traversing](#)

`.addBack( [selector ] )`

Returns: [jQuery](#)

**Description:** *Add the previous set of elements on the stack to the current set, optionally filtered by a selector.*

`.addBack( [selector ] )`

version added: [1.8](#)

**selector**

Type: [Selector](#)

A string containing a selector expression to match the current set of elements against.

As described in the discussion for `.end()`, jQuery objects maintain an internal stack that keeps track of changes to the matched set of elements. When one of the DOM traversal methods is called, the new set of elements is pushed onto the stack. If the previous set of elements is desired as well, `.addBack()` can help.

Consider a page with a simple list on it:

```
1 | <ul>
2 |   <li>list item 1</li>
3 |   <li>list item 2</li>
4 |   <li class="third-item">list item 3</li>
5 |   <li>list item 4</li>
6 |   <li>list item 5</li>
7 | </ul>
```

The result of the following code is a red background behind items 3, 4 and 5:

```
1 | $('li.third-item').nextAll().addBack()
2 |   .css('background-color', 'red');
```

First, the initial selector locates item 3, initializing the stack with the set containing just this item. The call to `.nextAll()` then pushes the set of items 4 and 5 onto the stack. Finally, the `.addBack()` invocation merges these two sets together, creating a jQuery object that points to all three items in document order: `{[<li.third-item>,<li>,<li> ]}`.

## Example:

The `addBack()` method causes the previous set of DOM elements in the traversal stack to be added to the current set. In the first example, the top stack contains the set resulting from `find("p")`. In the second example, `addBack()` adds the previous set of elements on the stack — in this case `$("#div-after-addback")` — to the current set, selecting both the div and its enclosed paragraphs.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5   p, div { margin:5px; padding:5px; }
6   .border { border: 2px solid red; }
7   .background { background:yellow; }
8   .left, .right { width: 45%; float: left;}
9   .right { margin-left:3%; }
10  </style>
11  <script src="http://code.jquery.com/jquery-
12 </head>
13 <body>
14
15 <div class="left">
16   <p><strong>Before <code>addBack()</code></s
17   <div class="before-addback">
18     <p>First Paragraph</p>
19     <p>Second Paragraph</p>
20   </div>
21 </div>
22 <div class="right">
23   <p><strong>After <code>addBack()</code></st
24   <div class="after-addback">
```

```
25     <p>First Paragraph</p>
26     <p>Second Paragraph</p>
27 </div>
28 </div>
29
30 <script>
31 $("div.left, div.right").find("div, div > p").addClass(
32
33 // First Example
34 $("div.before-addback").find("p").addClass("background"
35
36 // Second Example
37 $("div.after-addback").find("p").addBack().addClass(
38 </script>
39
40 </body>
41 </html>
```

Demo

POWERED BYHERONOTE

A new version of this book is available!

---

# .addClass()

Categories: [Attributes](#) | [Manipulation](#) > [Class Attribute](#) | [CSS](#)

`.addClass( className )`

Returns: [jQuery](#)

**Description:** *Adds the specified class(es) to each of the set of matched elements.*

**.addClass( className )**

version added: [1.0](#)

**className**

Type: [String](#)

One or more space-separated classes to be added to the class attribute of each matched element.

**.addClass( function(index, currentClass) )**

version added: [1.4](#)

**function(index, currentClass)**

Type: [Function\(\)](#)

A function returning one or more space-separated class names to be added to the existing class name(s). Receives the index position of the element in the set and the existing class name(s) as arguments. Within the function, `this` refers to the current element in the set.

It's important to note that this method does not replace a class. It simply adds the class, appending it to any which may already be assigned to the elements.

More than one class may be added at a time, separated by a space, to the set of matched elements, like so:

```
1 | $("p").addClass("myClass yourClass");
```

This method is often used with `.removeClass()` to switch elements' classes from one to another, like so:

```
1 | $("p").removeClass("myClass noClass").addClass
```

Here, the `myClass` and `noClass` classes are removed from all paragraphs, while `yourClass` is added.

As of jQuery 1.4, the `.addClass()` method's argument can receive a function.

```
1 | $("ul li:last").addClass(function(index) {  
2 |     return "item-" + index;  
3 | });
```

Given an unordered list with five `<li>` elements, this example adds the class "item-4" to the last `<li>`.

## Examples:

**Example:** Add the class "selected" to the matched elements.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     p { margin: 8px; font-size:16px; }
6     .selected { color:blue; }
7     .highlight { background:yellow; }
8   </style>
9   <script src="http://code.jquery.com/jquery-
10 </head>
11 <body>
12
13   <p>Hello</p>
14   <p>and</p>
15   <p>Goodbye</p>
16
17 <script>
18   $("p").last().addClass("selected");
19 </script>
20
21 </body>
22 </html>
```

Demo

**Example:** Add the classes "selected" and "highlight" to the matched elements.

```
1 <!DOCTYPE html>
```

```

2 <html>
3 <head>
4   <style>
5     p { margin: 8px; font-size:16px; }
6     .selected { color:red; }
7     .highlight { background:yellow; }
8   </style>
9   <script src="http://code.jquery.com/jquery-latest.js"
10 </head>
11 <body>
12   <p>Hello</p>
13   <p>and</p>
14   <p>Goodbye</p>
15 <script>
16   $("p:last").addClass("selected highlight");
17 </script>
18
19 </body>
20 </html>

```

## Demo

**Example:** Pass in a function to `addClass()` to add the "green" class to a div that already has a "red" class.

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     div { background: white; }
6     .red { background: red; }
7     .red.green { background: green; }
8   </style>
9   <script src="http://code.jquery.com/jquery-
10 </head>
11 <body>

```

```
12 <div>This div should be white</div>
13 <div class="red">This div will be green because it now
14     It would be red if the addClass function failed.
15 <div>This div should be white</div>
16 <p>There are zero green divs</p>
17
18
19 <script>
20     $("div").addClass(function(index, currentClass) {
21         var addedClass;
22
23         if ( currentClass === "red" ) {
24             addedClass = "green";
25             $("p").text("There is one green div");
26         }
27
28         return addedClass;
29     });
30 </script>
31
32 </body>
33 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# .after()

Categories: [Manipulation](#) > [DOM Insertion, Outside](#)

`.after( content [, content ] )`

Returns: [jQuery](#)

**Description:** *Insert content, specified by the parameter, after each element in the set of matched elements.*

`.after( content [, content ] )`

version added: [1.0](#)

**content**

Type: [htmlString](#) or [Element](#) or [jQuery](#)

HTML string, DOM element, or jQuery object to insert after each element in the set of matched elements.

**content**

Type: [htmlString](#) or [Element](#) or [Array](#) or [jQuery](#)

One or more additional DOM elements, arrays of elements, HTML strings, or jQuery objects to insert after each element in the set of matched elements.

`.after( function(index) )`

version added: [1.4](#)

**function(index)**

Type: [Function\(\)](#)

A function that returns an HTML string, DOM element(s), or jQuery object to insert after each element in the set of matched elements. Receives the index position of the element in the set as an argument. Within the function, `this` refers to the current element in the set.

The `.after()` and `.insertAfter()` methods perform the same task. The major difference is in the syntax—specifically, in the placement of the content and target. With `.after()`, the selector expression preceding the method is the container after which the content is inserted. With `.insertAfter()`, on the other hand, the content precedes the method, either as a selector expression or as markup created on the fly, and it is inserted after the target container.

Using the following HTML:

```
1 | <div class="container">
```

```
2 | <h2>Greetings</h2>
3 | <div class="inner">Hello</div>
4 | <div class="inner">Goodbye</div>
5 | </div>
```

Content can be created and then inserted after several elements at once:

```
1 | $(' .inner ').after( '<p>Test</p>' );
```

Each inner `<div>` element gets this new content:

```
1 | <div class="container">
2 |   <h2>Greetings</h2>
3 |   <div class="inner">Hello</div>
4 |   <p>Test</p>
5 |   <div class="inner">Goodbye</div>
6 |   <p>Test</p>
7 | </div>
```

An element in the DOM can also be selected and inserted after another element:

```
1 | $(' .container ').after($(' h2 '));
```

If an element selected this way is inserted into a single location elsewhere in the DOM, it will be moved rather than cloned:

```
1 | <div class="container">
2 |   <div class="inner">Hello</div>
3 |   <div class="inner">Goodbye</div>
4 | </div>
```

```
5 | <h2>Greetings</h2>
```

If there is more than one target element, however, cloned copies of the inserted element will be created for each target after the first.

### Inserting Disconnected DOM nodes

As of jQuery 1.4, `.before()` and `.after()` will also work on disconnected DOM nodes. For example, given the following code:

```
1 | $('<div/>').after('<p></p>');
```

The result is a jQuery set containing a div and a paragraph, in that order. That set can be further manipulated, even before it is inserted in the document.

```
1 | $('<div/>').after('<p></p>').addClass('foo')
2 |   .filter('p').attr('id', 'bar').html('hello')
3 |   .end()
4 |   .appendTo('body');
```

This results in the following elements inserted just before the closing `</body>` tag:

```
1 | <div class="foo"></div>
2 | <p class="foo" id="bar">hello</p>
```

### Passing a Function

As of jQuery 1.4, `.after()` supports passing a function that returns the elements to insert.

```
1 | $('p').after(function() {
```

```
2 | return '<div>' + this.className + '</div>';  
3 | });
```

This example inserts a `<div>` after each paragraph, with each new `<div>` containing the class name(s) of its preceding paragraph.

## Additional Arguments

Similar to other content-adding methods such as `.prepend()` and `.before()`, `.after()` also supports passing in multiple arguments as input. Supported input includes DOM elements, jQuery objects, HTML strings, and arrays of DOM elements.

For example, the following will insert two new `<div>`s and an existing `<div>` after the first paragraph:

```
1 | var $newdiv1 = $('<div id="object1"/>'),  
2 |     newdiv2 = document.createElement('div'),  
3 |     existingdiv1 = document.getElementById('f  
4 |  
5 | $('p').first().after($newdiv1, [newdiv2, exist
```

Since `.after()` can accept any number of additional arguments, the same result can be achieved by passing in the three `<div>`s as three separate arguments, like so: `$('p').first().after($newdiv1, newdiv2, existingdiv1)`. The type and number of arguments will largely depend on the elements that are collected in the code.

## Examples:

**Example:** *Inserts some HTML after all paragraphs.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <style>p { background:yellow; }</style>
5      <script src="http://code.jquery.com/jquery-
6  </head>
7  <body>
8      <p>I would like to say: </p>
9      <script>$("p").after("<b>Hello</b>");</script>
10
11 </body>
12 </html>
```

Demo

**Example:** *Inserts a DOM element after all paragraphs.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <style>p { background:yellow; }</style>
5      <script src="http://code.jquery.com/jquery-
6  </head>
7  <body>
8      <p>I would like to say: </p>
9      <script>$("p").after( document.createTextNode
10
11 </body>
12 </html>
```

## Demo

**Example** Inserts a jQuery object (similar to an Array of DOM Elements) after all paragraphs.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>p { background:yellow; }</style>
5   <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
6 </head>
7 <body>
8   <b>Hello</b><p>I would like to say: </p>
9   <script>$("p").after( $("b" ) );</script>
10
11 </body>
12 </html>
```

## Demo

POWERED BY HERONOTE

A new version of this book is available!

---

# .ajaxComplete()

Categories: [Ajax](#) > [Global Ajax Event Handlers](#)

`.ajaxComplete( handler(event, XMLHttpRequest, ajaxOptions) )` Returns: [jQuery](#)

**Description:** Register a handler to be called when Ajax requests complete. This is an [AjaxEvent](#).

`.ajaxComplete( handler(event, XMLHttpRequest, ajaxOptions) )` version added: [1.0](#)

**handler(event, XMLHttpRequest, ajaxOptions)**

Type: [Function\(\)](#)

The function to be invoked.

Whenever an Ajax request completes, jQuery triggers the `ajaxComplete` event. Any and all handlers that have been registered with the `.ajaxComplete()` method are executed at this time.

To observe this method in action, set up a basic Ajax load request:

```
1 | <div class="trigger">Trigger</div>
2 | <div class="result"></div>
3 | <div class="log"></div>
```

Attach the event handler to the document:

```
1 | $(document).ajaxComplete(function() {
2 |     $( ".log" ).text( "Triggered ajaxComplete h
3 | });
```

Now, make an Ajax request using any jQuery method:

```
1 | $( ".trigger" ).click(function() {
2 |     $( ".result" ).load( "ajax/test.html" );
```

```
3 | });
```

When the user clicks the element with class `trigger` and the Ajax request completes, the log message is displayed.

**Note:** Because `.ajaxComplete()` is implemented as a method of jQuery object instances, you can use the `this` keyword to refer to the selected elements within the callback function. **As of jQuery 1.8, however, the `.ajaxComplete()` method should only be attached to `document`.**

All `ajaxComplete` handlers are invoked, regardless of what Ajax request was completed. If you must differentiate between the requests, use the parameters passed to the handler. Each time an `ajaxComplete` handler is executed, it is passed the event object, the `XMLHttpRequest` object, and the settings object that was used in the creation of the request. For example, you can restrict the callback to only handling events dealing with a particular URL:

```
1 | $(document).ajaxComplete(function(event, xhr,
2 |     if ( settings.url === "ajax/test.html" ) {
3 |         $( ".log" ).text( "Triggered ajaxComplete
4 |                                 xhr.responseText );
5 |     }
6 | });
```

**Note:** You can get the returned ajax contents by looking at `xhr.responseXML` or `xhr.responseText` for xml and html respectively.

## Additional Notes:

If `$.ajax()` or `$.ajaxSetup()` is called with the `global` option set to `false`, the `.ajaxComplete()` method will not fire.

## Example:

*Show a message when an Ajax request completes.*

```
1 | $(document).ajaxComplete(function(event, request) {
2 |     $("#msg").append("<li>Request Complete.</li>");
3 | });
```

POWERED BY **HERONOTE**

A new version of this book is available!

---

# .ajaxError()

Categories: [Ajax](#) > [Global Ajax Event Handlers](#)

`.ajaxError( handler(event, jqXHR, ajaxSettings, errorThrown) )` Returns: [jQuery](#)

**Description:** Register a handler to be called when Ajax requests complete with an error. This is an [Ajax Event](#).

`.ajaxError( handler(event, jqXHR, ajaxSettings, errorThrown) )` version added: [1.0](#)

**handler(event, jqXHR, ajaxSettings, errorThrown)**

Type: [Function\(\)](#)

The function to be invoked.

Whenever an Ajax request completes with an error, jQuery triggers the `ajaxError` event. Any and all handlers that have been registered with the `.ajaxError()` method are executed at this time.

To observe this method in action, set up a basic Ajax load request.

```
1 | <button class="trigger">Trigger</button>
2 | <div class="result"></div>
3 | <div class="log"></div>
```

Attach the event handler to the document:

```
1 | $(document).ajaxError(function() {
2 |     $( "div.log" ).text( "Triggered ajaxError ha
3 | });
```

Now, make an Ajax request using any jQuery method:

```
1 | $( "button.trigger" ).click(function() {
2 |     $( "div.result" ).load( "ajax/missing.html"
```

```
3 | });
```

When the user clicks the button and the Ajax request fails, because the requested file is missing, the log message is displayed.

**Note:** Because `.ajaxError()` is implemented as a method of jQuery object instances, you can use the `this` keyword within the callback function to refer to the selected elements. **As of jQuery 1.8, however, the `.ajaxError()` method should only be attached to `document`.**

All `ajaxError` handlers are invoked, regardless of what Ajax request was completed. To differentiate between the requests, use the parameters passed to the handler. Each time an `ajaxError` handler is executed, it is passed the event object, the `jqXHR` object (prior to jQuery 1.5, the `XHR` object), and the settings object that was used in the creation of the request. If the request failed because JavaScript raised an exception, the exception object is passed to the handler as a fourth parameter. For example, to restrict the error callback to only handling events dealing with a particular URL:

```
1 | $( document ).ajaxError(function(event, jqxhr,  
2 |     if ( settings.url == "ajax/missing.html" )  
3 |         $( "div.log" ).text( "Triggered ajaxError  
4 |     }  
5 | });
```

## Additional Notes:

If `$.ajax()` or `$.ajaxSetup()` is called with the `global` option set to `false`, the `.ajaxError()` method will not fire.

## Example:

*Show a message when an Ajax request fails.*

```
1 | $(document).ajaxError(function(event, request,  
2 |     $("#msg").append("<li>Error requesting pa  
3 | });
```

POWERED BY **HERONOTE**

A new version of this book is available!

---

# .ajaxSend()

Categories: [Ajax](#) > [Global Ajax Event Handlers](#)

```
.ajaxSend( handler(event, jqXHR, ajaxOptions) )
```

Returns: [jQuery](#)

**Description:** Attach a function to be executed before an Ajax request is sent. This is an [Ajax Event](#).

```
.ajaxSend( handler(event, jqXHR,  
ajaxOptions) )
```

version added: [1.0](#)

**handler(event, jqXHR, ajaxOptions)**

Type: [Function\(\)](#)

The function to be invoked.

Whenever an Ajax request is about to be sent, jQuery triggers the `ajaxSend` event. Any and all handlers that have been registered with the `.ajaxSend()` method are executed at this time.

To observe this method in action, set up a basic Ajax load request:

```
1 | <div class="trigger">Trigger</div>  
2 | <div class="result"></div>  
3 | <div class="log"></div>
```

Attach the event handler to the document:

```
1 | $(document).ajaxSend(function() {  
2 |     $(".log").text("Triggered ajaxSend handle  
3 | });
```

Now, make an Ajax request using any jQuery method:

```
1 | $( ".trigger" ).click(function() {  
2 |     $( ".result" ).load( "ajax/test.html" );
```

```
3 | });
```

When the user clicks the element with class `trigger` and the Ajax request is about to begin, the log message is displayed.

**Note:** Because `.ajaxSend()` is implemented as a method of jQuery instances, you can use the `this` keyword to refer to the selected elements within the callback function. **As of jQuery 1.8, however, the `.ajaxSend()` method should only be attached to `document`.**

All `ajaxSend` handlers are invoked, regardless of what Ajax request is to be sent. If you must differentiate between the requests, use the parameters passed to the handler. Each time an `ajaxSend` handler is executed, it is passed the event object, the `jqXHR` object (in version 1.4, `XMLHttpRequest` object), and the [settings object](#) that was used in the creation of the Ajax request. For example, you can restrict the callback to only handling events dealing with a particular URL:

```
1 | $(document).ajaxSend(function(event, jqxhr, settings) {
2 |     if ( settings.url == "ajax/test.html" ) {
3 |         $( ".log" ).text( "Triggered ajaxSend handler" );
4 |     }
5 | });
```

## Additional Notes:

If `$.ajax()` or `$.ajaxSetup()` is called with the `global` option set to `false`, the `.ajaxSend()` method will not fire.

## Example:

*Show a message before an Ajax request is sent.*

```
1 | $(document).ajaxSend(function(event, request, XMLHttpRequest) {
2 |     $("#msg").append("<li>Starting request");
3 | });
```

POWERED BY HERONOTE

A new version of this book is available!

---

# .ajaxStart()

Categories: [Ajax](#) > [Global Ajax Event Handlers](#)

`.ajaxStart( handler() )`

Returns: [jQuery](#)

**Description:** Register a handler to be called when the first Ajax request begins. This is an [Ajax Event](#).

`.ajaxStart( handler() )`

version added: [1.0](#)

**handler()**

Type: [Function\(\)](#)

The function to be invoked.

Whenever an Ajax request is about to be sent, jQuery checks whether there are any other outstanding Ajax requests. If none are in progress, jQuery triggers the `ajaxStart` event. Any and all handlers that have been registered with the `.ajaxStart()` method are executed at this time.

To observe this method in action, set up a basic Ajax load request:

```
1 | <div class="trigger">Trigger</div>
2 | <div class="result"></div>
3 | <div class="log"></div>
```

Attach the event handler to any element:

```
1 | $(document).ajaxStart(function() {
2 |     $( ".log" ).text( "Triggered ajaxStart hand.
3 | });
```

Now, make an Ajax request using any jQuery method:

```
1 | $( ".trigger" ).click(function() {
2 |     $( ".result" ).load("ajax/test.html");
```

```
3 | });
```

When the user clicks the element with class `trigger` and the Ajax request is sent, the log message is displayed.

**Note:** Because `.ajaxStart()` is implemented as a method of jQuery object instances, you can use the `this` keyword to refer to the selected elements within the callback function. **As of jQuery 1.8, however, the `.ajaxStart()` method should only be attached to `document`.**

## Additional Notes:

If `$.ajax()` or `$.ajaxSetup()` is called with the `global` option set to `false`, the `.ajaxStart()` method will not fire.

## Example:

*Show a loading message whenever an Ajax request starts (and none is already active).*

```
1 | $(document).ajaxStart(function() {  
2 |     $("#loading").show();  
3 | });
```

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# .ajaxStop()

Categories: [Ajax](#) > [Global Ajax Event Handlers](#)

`.ajaxStop( handler() )`

Returns: [jQuery](#)

**Description:** Register a handler to be called when all Ajax requests have completed. This is an [Ajax Event](#).

`.ajaxStop( handler() )`

version added: [1.0](#)

**handler()**

Type: [Function\(\)](#)

The function to be invoked.

Whenever an Ajax request completes, jQuery checks whether there are any other outstanding Ajax requests. If none remain, jQuery triggers the `ajaxStop` event. Any and all handlers that have been registered with the `.ajaxStop()` method are executed at this time. The `ajaxStop` event is also triggered if the last outstanding Ajax request is cancelled by returning false within the `beforeSend` callback function.

To observe this method in action, set up a basic Ajax load request:

```
1 | <div class="trigger">Trigger</div>
2 | <div class="result"></div>
3 | <div class="log"></div>
```

Attach the event handler to the document:

```
1 | $( ".log" ).ajaxStop(function() {
2 |     $(this).text( "Triggered ajaxStop handler."
3 | });
```

Now, make an Ajax request using any jQuery method:

```
1 | $( ".trigger" ).click(function() {  
2 |     $( ".result" ).load( "ajax/test.html" );  
3 | });
```

When the user clicks the element with class `trigger` and the Ajax request completes, the log message is displayed.

Because `.ajaxStop()` is implemented as a method of jQuery object instances, you can use the `this` keyword to refer to the selected elements within the callback function. **As of jQuery 1.8, however, the `.ajaxStop()` method should only be attached to `document`.**

## Additional Notes:

If `$.ajax()` or `$.ajaxSetup()` is called with the `global` option set to `false`, the `.ajaxStop()` method will not fire.

## Example:

*Hide a loading message after all the Ajax requests have stopped.*

```
1 | $(document).ajaxStop(function() {  
2 |     $("#loading").hide();  
3 | });
```

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# .ajaxSuccess()

Categories: [Ajax](#) > [Global Ajax Event Handlers](#)

`.ajaxSuccess( handler(event, XMLHttpRequest, ajaxOptions) )` Returns: [jQuery](#)

**Description:** Attach a function to be executed whenever an Ajax request completes successfully. This is an [Ajax Event](#).

`.ajaxSuccess( handler(event, XMLHttpRequest, ajaxOptions) )` version added: [1.0](#)

**handler(event, XMLHttpRequest, ajaxOptions)**

Type: [Function\(\)](#)

The function to be invoked.

Whenever an Ajax request completes successfully, jQuery triggers the `ajaxSuccess` event. Any and all handlers that have been registered with the `.ajaxSuccess()` method are executed at this time.

To observe this method in action, set up a basic Ajax load request:

```
1 | <div class="trigger">Trigger</div>
2 | <div class="result"></div>
3 | <div class="log"></div>
```

Attach the event handler to any element:

```
1 | $(document).ajaxSuccess(function() {
2 |     $(".log").text("Triggered ajaxSuccess ha
3 | });
```

Now, make an Ajax request using any jQuery method:

```
1 | $(".trigger").on("click", function() {
```

```
2 | $( ".result" ).load( "ajax/test.html" );
3 | });
```

When the user clicks the element with class `trigger` and the Ajax request completes successfully, the log message is displayed.

**Note:** Because `.ajaxSuccess()` is implemented as a method of jQuery object instances, use the `this` keyword to refer to the selected elements within the callback function. **As of jQuery 1.8, however, the `.ajaxSuccess()` method should only be attached to `document`.**

All `ajaxSuccess` handlers are invoked, regardless of what Ajax request was completed. If you must differentiate between the requests, you can use the parameters passed to the handler. Each time an `ajaxSuccess` handler is executed, it is passed the event object, the `XMLHttpRequest` object, and the settings object that was used in the creation of the request. For example, you can restrict the callback to only handling events dealing with a particular URL:

```
1 | $(document).ajaxSuccess(function(event, xhr, s
2 |     if ( settings.url == "ajax/test.html" ) {
3 |         $( ".log" ).text( "Triggered ajaxSuccess l
4 |                                 xhr.responseText );
5 |     }
6 | });
```

**Note:** You can get the returned ajax contents by looking at `xhr.responseXML` or `xhr.responseText` for xml and html respectively.

## Additional Notes:

If `$.ajax()` or `$.ajaxSetup()` is called with the `global` option set to `false`, the `.ajaxSuccess()` method will not fire.

## Example:

*Show a message when an Ajax request completes successfully.*

```
1 | $(document).ajaxSuccess(function(event, request, response) {
2 |     $("#msg").append("<li>Successful Request");
3 | });
```

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# All Selector (“\*”)

Categories: [Selectors](#) > [Basic](#)

## all selector

**Description:** *Selects all elements.*

**jQuery( "\*" )**

version added: [1.0](#)

Caution: The all, or universal, selector is extremely slow, except when used by itself.

## Examples:

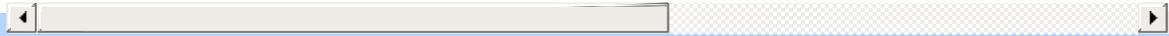
**Example:** Find every element (including head, body, etc) in the document. Note that if your browser has an extension/add-on enabled that inserts a `<script>` or `<link>` element into the DOM, that element will be counted as well.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     h3 { margin: 0; }
6     div, span, p {
7       width: 80px;
8       height: 40px;
9       float:left;
10      padding: 10px;
11      margin: 10px;
12      background-color: #EEEEEE;
13    }
14  </style>
15  <script src="http://code.jquery.com/jquery-
16 </head>
17 <body>
18   <div>DIV</div>
19
20   <span>SPAN</span>
21   <p>P <button>Button</button></p>
22   <script>var elementCount = $("*").css("border
23   $("body").prepend("<h3>" + elementCount + " e.
24
25 </body>
26 </html>
```

## Demo

**Example:** Find all elements within document body so elements like head, script, etc. are excluded.

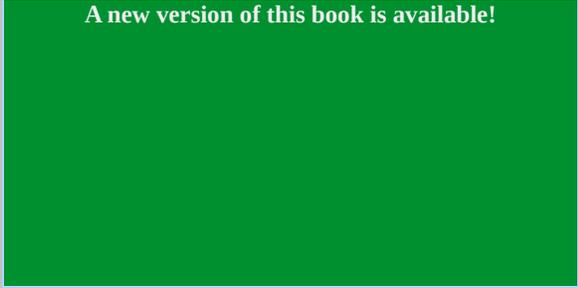
```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5      h3 { margin: 0; }
6      div,span,p {
7        width: 80px;
8        height: 40px;
9        float:left;
10       padding: 10px;
11       margin: 10px;
12       background-color: #EEEEEE;
13     }
14     #test {
15       width: auto; height: auto; background-color: #EEEEEE;
16     }
17   </style>
18   <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
19 </head>
20 <body>
21   <div id="test">
22     <div>DIV</div>
23     <span>SPAN</span>
24     <p>P <button>Button</button></p>
25   </div>
26   <script>
27     var elementCount = $("#test").find("*").css("background-color");
28     $("body").prepend("<h3>" + elementCount + " elements found.");
29   </script>
30 </body>
31 </html>
```



Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

# .andSelf()

Categories: [Traversing](#) > [Miscellaneous Traversing](#)

`.andSelf()`

Returns: [jQuery](#)  
version deprecated: [1.8](#)

**Description:** *Add the previous set of elements on the stack to the current set.*

`.andSelf()`

version added: [1.2](#)

This method does not accept any arguments.

**Note:** This function has been deprecated and is now an alias for `.addBack()`, which should be used with jQuery 1.8 and later.

As described in the discussion for `.end()`, jQuery objects maintain an internal stack that keeps track of changes to the matched set of elements. When one of the DOM traversal methods is called, the new set of elements is pushed onto the stack. If the previous set of elements is desired as well, `.andSelf()` can help.

Consider a page with a simple list on it:

```
1 | <ul>
2 |   <li>list item 1</li>
3 |   <li>list item 2</li>
4 |   <li class="third-item">list item 3</li>
5 |   <li>list item 4</li>
6 |   <li>list item 5</li>
7 | </ul>
```

The result of the following code is a red background behind items 3, 4 and 5:

```
1 | $('li.third-item').nextAll().andSelf()
2 |   .css('background-color', 'red');
```

First, the initial selector locates item 3, initializing the stack with the set containing just this item. The call to `.nextAll()` then pushes the set of items 4 and 5 onto the stack. Finally, the `.andSelf()` invocation merges these two sets together, creating a jQuery object that points to all three items in document order: `{[<li.third-item>,<li>,<li> ]}`.

## Example:

The `andSelf()` method causes the previous set of DOM elements in the traversal stack to be added to the current set. In the first example, the top stack contains the set resulting from `find("p")`. In the second example, `andSelf()` adds the previous set of elements on the stack — in this case `$("#div-after-andself")` — to the current set, selecting both the div and its enclosed paragraphs.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5   p, div { margin:5px; padding:5px; }
6   .border { border: 2px solid red; }
7   .background { background:yellow; }
8   .left, .right { width: 45%; float: left;}
9   .right { margin-left:3%; }
10  </style>
11  <script src="http://code.jquery.com/jquery-
12 </head>
13 <body>
14
15 <div class="left">
16   <p><strong>Before <code>andSelf()</code></s
17   <div class="before-andself">
18     <p>First Paragraph</p>
19     <p>Second Paragraph</p>
20   </div>
21 </div>
22 <div class="right">
23   <p><strong>After <code>andSelf()</code></st
24   <div class="after-andself">
```

```
25     <p>First Paragraph</p>
26     <p>Second Paragraph</p>
27 </div>
28 </div>
29
30 <script>
31 $("div.left, div.right").find("div, div > p").addClass(
32
33 // First Example
34 $("div.before-andself").find("p").addClass("background"
35
36 // Second Example
37 $("div.after-andself").find("p").andSelf().addClass(
38 </script>
39
40 </body>
41 </html>
```

Demo

POWERED BYHERONOTE

A new version of this book is available!

---

# .animate()

Categories: [Effects](#) > [Custom](#)

`.animate( properties [, duration ] [, easing ] [, complete ] )`

Returns: [jQuery](#)

**Description:** *Perform a custom animation of a set of CSS properties.*

`.animate( properties [, duration ] [, easing ] [, complete ] )`

version added: [1.0](#)

#### **properties**

Type: [PlainObject](#)

An object of CSS properties and values that the animation will move toward.

#### **duration** (default: `400`)

Type: [Number](#) or [String](#)

A string or number determining how long the animation will run.

#### **easing** (default: `swing`)

Type: [String](#)

A string indicating which easing function to use for the transition.

#### **complete**

Type: [Function\(\)](#)

A function to call once the animation is complete.

`.animate( properties, options )`

version added: [1.0](#)

#### **properties**

Type: [PlainObject](#)

An object of CSS properties and values that the animation will move toward.

#### **options**

Type: [PlainObject](#)

A map of additional options to pass to the method.

#### **duration** (default: `400`)

Type: [Number](#) or [String](#)

A string or number determining how long the animation will

run.

**easing** (default: `swing`)

Type: [String](#)

A string indicating which easing function to use for the transition.

**queue** (default: `true`)

Type: [Boolean](#)

A Boolean indicating whether to place the animation in the effects queue. If false, the animation will begin immediately.

**As of jQuery 1.7**, the queue option can also accept a string, in which case the animation is added to the queue represented by that string.

**specialEasing**

Type: [PlainObject](#)

A map of one or more of the CSS properties defined by the properties argument and their corresponding easing functions. **(version added: 1.4)**

**step**

Type: [Function](#)( [Number](#) now, [Tween](#) tween )

A function to be called for each animated property of each animated element. This function provides an opportunity to modify the Tween object to change the value of the property before it is set.

**progress**

Type: [Function](#)( [Promise](#) animation, [Number](#) progress, [Number](#) remainingMs )

A function to be called after each step of the animation, only once per animated element regardless of the number of animated properties. **(version added: 1.8)**

**complete**

Type: [Function](#)()

A function to call once the animation is complete.

**done**

Type: [Function](#)( [Promise](#) animation, [Boolean](#) jumpedToEnd )

A function to be called when the animation completes (its Promise object is resolved). **(version added: 1.8)**

**fail**

Type: [Function](#)( [Promise](#) animation, [Boolean](#) jumpedToEnd )

A function to be called when the animation fails to complete (its Promise object is rejected). **(version added: [1.8](#))**

**always**

Type: [Function](#)( [Promise](#) animation, [Boolean](#) jumpedToEnd )

A function to be called when the animation completes or stops without completing (its Promise object is either resolved or rejected). **(version added: [1.8](#))**

The `.animate()` method allows us to create animation effects on any numeric CSS property. The only required parameter is a plain object of CSS properties. This object is similar to the one that can be sent to the `.css()` method, except that the range of properties is more restrictive.

### Animation Properties and Values

All animated properties should be animated to a *single numeric value*, except as noted below; most properties that are non-numeric cannot be animated using basic jQuery functionality (For example, `width`, `height`, or `left` can be animated but `background-color` cannot be, unless the [jQuery.Color\(\)](#) plugin is used). Property values are treated as a number of pixels unless otherwise specified. The units `em` and `%` can be specified where applicable.

In addition to style properties, some non-style properties such as `scrollTop` and `scrollLeft`, as well as custom properties, can be animated.

Shorthand CSS properties (e.g. `font`, `background`, `border`) are not fully supported. For example, if you want to animate the rendered border width, at least a border style and border width other than "auto" must be set in advance. Or, if you want to animate font size, you would use `fontSize` or the CSS equivalent `'font-size'` rather than simply `'font'`.

In addition to numeric values, each property can take the strings `'show'`, `'hide'`, and `'toggle'`. These shortcuts allow for custom

hiding and showing animations that take into account the display type of the element.

Animated properties can also be relative. If a value is supplied with a leading `+=` or `-=` sequence of characters, then the target value is computed by adding or subtracting the given number from the current value of the property.

**Note:** Unlike shorthand animation methods such as `.slideDown()` and `.fadeIn()`, the `.animate()` method does **not** make hidden elements visible as part of the effect. For example, given `$('#someElement').hide().animate({height:'20px'}, 500)`, the animation will run, but **the element will remain hidden**.

## Duration

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The default duration is `400` milliseconds. The strings `'fast'` and `'slow'` can be supplied to indicate durations of `200` and `600` milliseconds, respectively.

## Complete Function

If supplied, the `complete` callback function is fired once the animation is complete. This can be useful for stringing different animations together in sequence. The callback is not sent any arguments, but `this` is set to the DOM element being animated. If multiple elements are animated, the callback is executed once per matched element, not once for the animation as a whole.

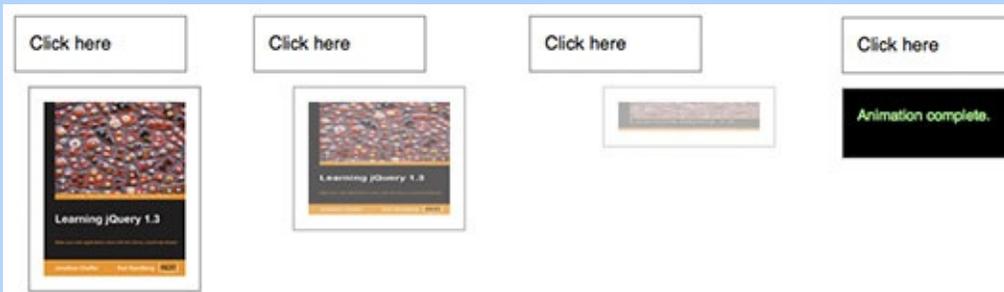
## Basic Usage

To animate any element, such as a simple image:

```
1 <div id="clickme">
2   Click here
3 </div>
4 
```

To animate the opacity, left offset, and height of the image simultaneously:

```
1 $( '#clickme' ).click(function() {
2   $( '#book' ).animate({
3     opacity: 0.25,
4     left: '+=50',
5     height: 'toggle'
6   }, 5000, function() {
7     // Animation complete.
8   });
9 });
```



Note that the target value of the `height` property is `'toggle'`. Since the image was visible before, the animation shrinks the height to 0 to hide it. A second click then reverses this transition:



The `opacity` of the image is already at its target value, so this property is not animated by the second click. Since the target value for `left` is a relative value, the image moves even farther to the right during this second animation.

Directional properties (`top`, `right`, `bottom`, `left`) have no discernible effect on elements if their `position` style property is `static`, which it is by default.

**Note:** The [jQuery UI](#) project extends the `.animate()` method by allowing some non-numeric styles such as colors to be animated. The project also includes mechanisms for specifying animations through CSS classes rather than individual attributes.

**Note:** if attempting to animate an element with a height or width of `0px`, where contents of the element are visible due to overflow, jQuery may clip this overflow during animation. By fixing the dimensions of the original element being hidden

however, it is possible to ensure that the animation runs smoothly. A [clearfix](#) can be used to automatically fix the dimensions of your main element without the need to set this manually.

## Step Function

The second version of `.animate()` provides a `step` option — a callback function that is fired at each step of the animation. This function is useful for enabling custom animation types or altering the animation as it is occurring. It accepts two arguments (`now` and `fx`), and `this` is set to the DOM element being animated.

`now`: the numeric value of the property being animated at each step

`fx`: a reference to the `jQuery.fx` prototype object, which contains a number of properties such as `elem` for the animated element, `start` and `end` for the first and last value of the animated property, respectively, and `prop` for the property being animated.

Note that the `step` function is called for each animated property on each animated element. For example, given two list items, the `step` function fires four times at each step of the animation:

```
1 | $('li').animate({
2 |   opacity: .5,
3 |   height: '50%'
4 | },
5 | {
6 |   step: function(now, fx) {
7 |     var data = fx.elem.id + ' ' + fx.prop + ' ' + now;
8 |     $('body').append('<div>' + data + '</div>');
9 |   }
}
```

```
10 | });
```

## Easing

The remaining parameter of `.animate()` is a string naming an easing function to use. An easing function specifies the speed at which the animation progresses at different points within the animation. The only easing implementations in the jQuery library are the default, called `swing`, and one that progresses at a constant pace, called `linear`. More easing functions are available with the use of plug-ins, most notably the [jQuery UI suite](#).

## Per-property Easing

As of jQuery version 1.4, you can set per-property easing functions within a single `.animate()` call. In the first version of `.animate()`, each property can take an array as its value: The first member of the array is the CSS property and the second member is an easing function. If a per-property easing function is not defined for a particular property, it uses the value of the `.animate()` method's optional easing argument. If the easing argument is not defined, the default `swing` function is used.

For example, to simultaneously animate the width and height with the `swing` easing function and the opacity with the `linear` easing function:

```
1 | $('#clickme').click(function() {
2 |     $('#book').animate({
3 |         width: ['toggle', 'swing'],
4 |         height: ['toggle', 'swing'],
5 |         opacity: 'toggle'
6 |     }, 5000, 'linear', function() {
7 |         $(this).after('<div>Animation complete.</div>');
8 |     });
9 | });
```

In the second version of `.animate()`, the options object can include the `specialEasing` property, which is itself an object of CSS properties and their corresponding easing functions. For example, to simultaneously animate the width using the `linear` easing function and the height using the `easeOutBounce` easing function:

```
1  $('#clickme').click(function() {
2    $('#book').animate({
3      width: 'toggle',
4      height: 'toggle'
5    }, {
6      duration: 5000,
7      specialEasing: {
8        width: 'linear',
9        height: 'easeOutBounce'
10     },
11     complete: function() {
12       $(this).after('<div>Animation complete
13     }
14   });
15 });
```

As previously noted, a plugin is required for the `easeOutBounce` function.

## Additional Notes:

All jQuery effects, including `.animate()`, can be turned off globally by setting `jQuery.fx.off = true`, which effectively sets the duration to 0. For more information, see [jQuery.fx.off](#).

## Examples:

**Example:** Click the button to animate the div with a number of different properties.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5   div {
6     background-color:#bca;
7     width:100px;
8     border:1px solid green;
9   }
10 </style>
11   <script src="http://code.jquery.com/jquery-
12 </head>
13 <body>
14   <button id="go">&raquo; Run</button>
15
16   <div id="block">Hello!</div>
17 <script>
18
19   /* Using multiple unit types within one anima
20
21   $( "#go" ).click( function() {
22     $( "#block" ).animate( {
23       width: "70%",
24       opacity: 0.4,
25       marginLeft: "0.6in",
26       fontSize: "3em",
27       borderWidth: "10px"
28     }, 1500 );
29   });
30 </script>
```

```
31  
32 </body>  
33 </html>
```

## Demo

**Example:** *Animates a div's left property with a relative value. Click several times on the buttons to see the relative animations queued up.*

```
1 <!DOCTYPE html>  
2 <html>  
3 <head>  
4 <style>  
5 div {  
6 position: absolute;  
7 background-color: #abc;  
8 left: 50px;  
9 width: 90px;  
10 height: 90px;  
11 margin: 5px;  
12 }  
13 </style>  
14 <script src="http://code.jquery.com/jquery-  
15 </head>  
16 <body>  
17 <button id="left">&laquo;</button> <button  
18 <div class="block"></div>  
19  
20 <script>  
21 $("#right").click(function(){  
22 $(".block").animate({"left": "+=50px"}, "sl  
23 });  
24  
25 $("#left").click(function(){  
26 $(".block").animate({"left": "-=50px"}, "sl
```

```
27 | });
28 |
29 | </script>
30 |
31 | </body>
32 | </html>
```

## Demo

**Example:** *The first button shows how an unqueued animation works. It expands the div out to 90% width while the font-size is increasing. Once the font-size change is complete, the border animation will begin. The second button starts a traditional chained animation where each animation will start once the previous animation on the element has completed.*

```
1 | <!DOCTYPE html>
2 | <html>
3 | <head>
4 |   <style>
5 |     div {
6 |       background-color:#bca;
7 |       width:200px;
8 |       height:1.1em;
9 |       text-align:center;
10 |      border:2px solid green;
11 |      margin:3px;
12 |      font-size:14px;
13 |    }
14 |    button {
15 |      font-size:14px;
16 |    }
17 |  </style>
18 |    <script src="http://code.jquery.com/jquery-
19 |  </head>
20 |  <body>
```

```

21 <button id="go1">&raquo; Animate Block1</button>
22 <button id="go2">&raquo; Animate Block2</button>
23 <button id="go3">&raquo; Animate Both</button>
24
25 <button id="go4">&raquo; Reset</button>
26 <div id="block1">Block1</div>
27 <div id="block2">Block2</div>
28 <script>
29
30 $( "#go1" ).click(function(){
31     $( "#block1" ).animate( { width: "90%" }, { queue:
32         .animate({ fontSize: "24px" }, 1500 )
33         .animate({ borderRightWidth: "15px" }, 1500
34     });
35
36 $( "#go2" ).click(function(){
37     $( "#block2" ).animate({ width: "90%" }, 1000
38         .animate({ fontSize: "24px" }, 1000 )
39         .animate({ borderLeftWidth: "15px" }, 1000
40     });
41
42 $( "#go3" ).click(function(){
43     $( "#go1" ).add( "#go2" ).click();
44 });
45
46 $( "#go4" ).click(function(){
47     $( "div" ).css({ width: "", fontSize: "", borderWidth
48 });
49
50 </script>
51
52 </body>
53 </html>

```

Demo

**Example:** *Animates the first div's left property and*

*synchronizes the remaining divs using the step function to set their left properties at each stage of the animation.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5   div {
6     position: relative;
7     background-color: #abc;
8     width: 40px;
9     height: 40px;
10    float: left;
11    margin: 5px;
12  }
13 </style>
14   <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
15 </head>
16 <body>
17
18 <p><button id="go">Run >></button></p>
19 <div class="block"></div> <div class="block"></div>
20 <div class="block"></div> <div class="block"></div>
21 <div class="block"></div> <div class="block"></div>
22
23 <script>
24 $( "#go" ).click(function(){
25   $( ".block:first" ).animate({
26     left: 100
27   }, {
28     duration: 1000,
29     step: function( now, fx ){
30       $( ".block:gt(0)" ).css( "left", now );
31     }
32   });

```

```
33 | });  
34 | </script>  
35 |  
36 | </body>  
37 | </html>
```

## Demo

**Example:** *Animate all paragraphs to toggle both height and opacity completing the animation within 600 milliseconds.*

```
1 | $( "p" ).animate({  
2 |   height: "toggle", opacity: "toggle"  
3 | }, "slow" );
```

**Example:** *Animate all paragraphs to a left style of 50 and opacity of 1 (opaque visible) completing the animation within 500 milliseconds.*

```
1 | $( "p" ).animate({  
2 |   left: 50, opacity: 1  
3 | }, 500 );
```

**Example:** *Animate the left and opacity style properties of all paragraphs: run the animation outside the queue, so that it will automatically start without waiting for its turn.*

```
1 | $( "p" ).animate({  
2 |   left: "50px", opacity: 1  
3 | }, { duration: 500, queue: false });
```

**Example:** An example of using an 'easing' function to provide a different style of animation. This will only work if you have a plugin that provides this easing function. Note this code will do nothing unless the paragraph element is hidden.

```
1 | $( "p" ).animate({
2 |   opacity: "show"
3 | }, "slow", "easein" );
```

**Example:** Animates all paragraphs to toggle both height and opacity, completing the animation within 600 milliseconds.

```
1 | $( "p" ).animate({
2 |   height: "toggle", opacity: "toggle"
3 | }, { duration: "slow" });
```

**Example:** Use an easing function to provide a different style of animation. This will only work if you have a plugin that provides this easing function.

```
1 | $( "p" ).animate({
2 |   opacity: "show"
3 | }, { duration: "slow", easing: "easein" });
```

**Example:** Animate all paragraphs and execute a callback function when the animation is complete. The first argument is an object of CSS properties, the second specifies that the animation should take 1000 milliseconds to complete, the third states the easing type, and the fourth argument is an anonymous callback function.

```
1 | $( "p" ).animate({
2 |     height: 200, width: 400, opacity: 0.5
3 | }, 1000, "linear", function() {
4 |     alert("all done");
5 | });
```

POWERED BY HERONOTE

A new version of this book is available!

---

# :animated Selector

Categories: [Selectors](#) > [Basic Filter](#) | [Selectors](#) > [jQuery Extensions](#)

## animated selector

**Description:** *Select all elements that are in the progress of an animation at the time the selector is run.*

**jQuery( ":animated" )**

version added: [1.2](#)

**Note:** If you use a custom jQuery build *without the effects module*, the `:animated` selector will throw an error.

### Additional Notes:

Because `:animated` is a jQuery extension and not part of the CSS specification, queries using `:animated` cannot take advantage of the performance boost provided by the native DOM `querySelectorAll()` method. To achieve the best performance when using `:animated` to select elements, first select the elements using a pure CSS selector, then use `.filter(":animated")`.

## Example:

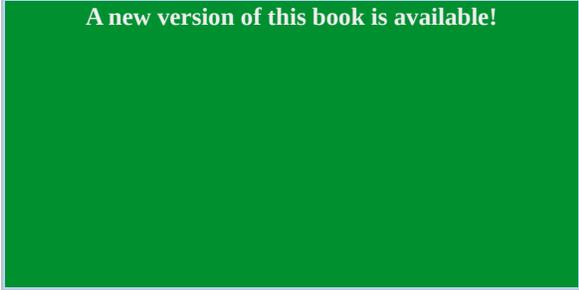
*Change the color of any div that is animated.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5    div { background:yellow; border:1px solid #;
6    div.colored { background:green; }
7    </style>
8    <script src="http://code.jquery.com/jquery-
9  </head>
10 <body>
11   <button id="run">Run</button>
12
13   <div></div>
14   <div id="mover"></div>
15   <div></div>
16 <script>
17
18   $("#run").click(function(){
19     $("div:animated").toggleClass("colored"
20   });
21   function animateIt() {
22     $("#mover").slideToggle("slow", animate
23   }
24   animateIt();
25 </script>
26
27 </body>
28 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

# .append()

Categories: [Manipulation](#) > [DOM Insertion, Inside](#)

`.append( content [, content ] )`

Returns: [jQuery](#)

**Description:** *Insert content, specified by the parameter, to the end of each element in the set of matched elements.*

`.append( content [, content ] )`

version added: [1.0](#)

**content**

Type: [htmlString](#) or [Element](#) or [jQuery](#)

DOM element, HTML string, or jQuery object to insert at the end of each element in the set of matched elements.

**content**

Type: [htmlString](#) or [Element](#) or [Array](#) or [jQuery](#)

One or more additional DOM elements, arrays of elements, HTML strings, or jQuery objects to insert at the end of each element in the set of matched elements.

`.append( function(index, html) )`

version added: [1.4](#)

**function(index, html)**

Type: [Function\(\)](#)

A function that returns an HTML string, DOM element(s), or jQuery object to insert at the end of each element in the set of matched elements. Receives the index position of the element in the set and the old HTML value of the element as arguments. Within the function, `this` refers to the current element in the set.

The `.append()` method inserts the specified content as the last child of each element in the jQuery collection (To insert it as the *first* child, use `.prepend()`).

The `.append()` and `.appendTo()` methods perform the same task. The major difference is in the syntax—specifically, in the placement of the content and target. With `.append()`, the selector expression preceding the method is the container into which the content is inserted. With `.appendTo()`, on the other hand, the content precedes the method, either as a selector expression or as markup created on

the fly, and it is inserted into the target container.

Consider the following HTML:

```
1 | <h2>Greetings</h2>
2 | <div class="container">
3 |   <div class="inner">Hello</div>
4 |   <div class="inner">Goodbye</div>
5 | </div>
```

You can create content and insert it into several elements at once:

```
1 | $(' .inner ' ).append( '<p>Test</p>' );
```

Each inner `<div>` element gets this new content:

```
1 | <h2>Greetings</h2>
2 | <div class="container">
3 |   <div class="inner">
4 |     Hello
5 |     <p>Test</p>
6 |   </div>
7 |   <div class="inner">
8 |     Goodbye
9 |     <p>Test</p>
10 |   </div>
11 | </div>
```

You can also select an element on the page and insert it into another:

```
1 | $(' .container ' ).append($(' h2 ' ));
```

If an element selected this way is inserted into a single location elsewhere in the DOM, it will be moved into the target (not cloned):

```
1 | <div class="container">
2 |   <div class="inner">Hello</div>
3 |   <div class="inner">Goodbye</div>
4 |   <h2>Greetings</h2>
5 | </div>
```

If there is more than one target element, however, cloned copies of the inserted element will be created for each target after the first.

### Additional Arguments

Similar to other content-adding methods such as `.prepend()` and `.before()`, `.append()` also supports passing in multiple arguments as input. Supported input includes DOM elements, jQuery objects, HTML strings, and arrays of DOM elements.

For example, the following will insert two new `<div>`s and an existing `<div>` as the last three child nodes of the body:

```
1 | var $newdiv1 = $('<div id="object1"/>'),
2 |     newdiv2 = document.createElement('div'),
3 |     existingdiv1 = document.getElementById('f
4 |
5 | $('body').append($newdiv1, [newdiv2, existingdiv1]);
```

Since `.append()` can accept any number of additional arguments, the same result can be achieved by passing in the three `<div>`s as three separate arguments, like so: `$('body').append($newdiv1, newdiv2, existingdiv1)`. The type and number of arguments will largely depend on how you collect the elements in your code.

## Examples:

**Example:** *Appends some HTML to all paragraphs.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5      p { background:yellow; }
6    </style>
7    <script src="http://code.jquery.com/jquery-
8  </head>
9  <body>
10   <p>I would like to say: </p>
11   <script>
12     $("p").append("<strong>Hello</strong>");
13   </script>
14
15  </body>
16  </html>
```

Demo

**Example:** *Appends an Element to all paragraphs.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5      p { background:yellow; }
6    </style>
7    <script src="http://code.jquery.com/jquery-
8  </head>
9  <body>
```

```
10 <p>I would like to say: </p>
11
12 <script>
13   $("p").append(document.createTextNode("Hello"))
14 </script>
15
16 </body>
17 </html>
```

Demo

**Example:** Appends a jQuery object (similar to an Array of DOM Elements) to all paragraphs.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     p { background:yellow; }
6   </style>
7   <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
8 </head>
9 <body>
10  <strong>Hello world!!!</strong><p>I would like to say: </p>
11 <script>
12   $("p").append( $("strong") );
13 </script>
14
15 </body>
16 </html>
```

Demo

**A new version of this book is available!**

---

# .appendTo()

Categories: [Manipulation](#) > [DOM Insertion, Inside](#)

`.appendTo( target )`

Returns: `jQuery`

**Description:** *Insert every element in the set of matched elements to the end of the target.*

**.appendTo( target )**

version added: [1.0](#)

**target**

Type: [Selector](#) or [htmlString](#) or [Element](#) or [jQuery](#)

A selector, element, HTML string, or jQuery object; the matched set of elements will be inserted at the end of the element(s) specified by this parameter.

The `.append()` and `.appendTo()` methods perform the same task. The major difference is in the syntax-specifically, in the placement of the content and target. With `.append()`, the selector expression preceding the method is the container into which the content is inserted. With `.appendTo()`, on the other hand, the content precedes the method, either as a selector expression or as markup created on the fly, and it is inserted into the target container.

Consider the following HTML:

```
1 | <h2>Greetings</h2>
2 | <div class="container">
3 |   <div class="inner">Hello</div>
4 |   <div class="inner">Goodbye</div>
5 | </div>
```

We can create content and insert it into several elements at once:

```
1 | $( '<p>Test</p>' ).appendTo( '.inner' );
```

Each inner `<div>` element gets this new content:

```
1 | <h2>Greetings</h2>
2 | <div class="container">
3 |   <div class="inner">
4 |     Hello
5 |     <p>Test</p>
6 |   </div>
7 |   <div class="inner">
8 |     Goodbye
9 |     <p>Test</p>
10 |   </div>
11 | </div>
```

We can also select an element on the page and insert it into another:

```
1 | $('h2').appendTo($('.container'));
```

If an element selected this way is inserted into a single location elsewhere in the DOM, it will be moved into the target (not cloned):

```
1 | <div class="container">
2 |   <div class="inner">Hello</div>
3 |   <div class="inner">Goodbye</div>
4 |   <h2>Greetings</h2>
5 | </div>
```

If there is more than one target element, however, cloned copies of the inserted element will be created for each target after the first, and that new set (the original element plus clones) is returned.

## Example:

*Append all spans to the element with the ID "foo"  
(Check `append()` documentation for more examples)*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <style>#foo { background:yellow; }</style>
5      <script src="http://code.jquery.com/jquery-
6  </head>
7  <body>
8      <span>I have nothing more to say... </span>
9
10     <div id="foo">FOO! </div>
11     <script>
12     $("span").appendTo("#foo");
13     </script>
14
15     </body>
16     </html>
```

Demo

POWERED BYHERONOTE

A new version of this book is available!

# .attr()

Categories: [Attributes](#) | [Manipulation](#) > [General Attributes](#)

Get the value of an attribute for the first element in the set of matched elements or set one or more attributes for every matched element.

## Contents:

[.attr\( attributeName \)](#)

`.attr( attributeName )`

[.attr\( attributeName, value \)](#)

`.attr( attributeName, value )`

`.attr( attributes )`

`.attr( attributeName, function(index, attr) )`

`.attr( attributeName )`

Returns: [String](#)

**Description:** *Get the value of an attribute for the first element in the set of matched elements.*

`.attr( attributeName )`

version added: [1.0](#)

**attributeName**

Type: [String](#)

The name of the attribute to get.

The `.attr()` method gets the attribute value for only the *first* element in the matched set. To get the value for each element individually, use a looping construct such as jQuery's `.each()` or `.map()` method.

**As of jQuery 1.6**, the `.attr()` method returns `undefined` for attributes that have not been set. In addition, `.attr()` should not be used on plain objects, arrays, the window, or the document. To retrieve and change DOM properties, use the `.prop()` method.

Using jQuery's `.attr()` method to get the value of an element's attribute has two main benefits:

- 1 **Convenience:** It can be called directly on a jQuery object and chained to other jQuery methods.
- 2 **Cross-browser consistency:** The values of some attributes are reported inconsistently across browsers, and even across versions of a single browser. The `.attr()` method reduces such inconsistencies.

**Note:** *Attribute values are strings with the exception of a few attributes such as `value` and `tabindex`.*

## Example:

*Find the title attribute of the first <em> in the page.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5      em { color:blue; font-weight:bold; }
6      div { color:red; }
7    </style>
8    <script src="http://code.jquery.com/jquery-
9  </head>
10 <body>
11
12 <p>
13   Once there was a <em title="huge, gigantic">
14 </p>
15
16   The title of the emphasis is:<div></div>
17
18 <script>
19 var title = $("em").attr("title");
20   $("div").text(title);
21 </script>
22
23 </body>
24 </html>
```

Demo

`.attr( attributeName, value )`

Returns: [jQuery](#)

**Description:** *Set one or more attributes for the set of matched elements.*

**.attr( attributeName, value )**

version added: [1.0](#)

**attributeName**

Type: [String](#)

The name of the attribute to set.

**value**

Type: [String](#) or [Number](#)

A value to set for the attribute.

**.attr( attributes )**

version added: [1.0](#)

**attributes**

Type: [PlainObject](#)

An object of attribute-value pairs to set.

**.attr( attributeName, function(index, attr) )**

version added: [1.1](#)

**attributeName**

Type: [String](#)

The name of the attribute to set.

**function(index, attr)**

Type: [Function\(\)](#)

A function returning the value to set. `this` is the current element. Receives the index position of the element in the set and the old attribute value as arguments.

The `.attr()` method is a convenient way to set the value of attributes—especially when setting multiple attributes or using values returned by a function. Consider the following image:

```
1 | ` or `<button>` element and will throw an error in all browsers. This is because the `type` attribute cannot be changed in Internet Explorer.

## Computed attribute values

By using a function to set attributes, you can compute the value based on other properties of the element. For example, to concatenate a new value with an existing value:

```
1 | $('#greatphoto').attr('title', function(i, val)
2 |     return val + ' - photo by Kelly Clark'
3 | });
```

This use of a function to compute attribute values can be particularly useful when modifying the attributes of multiple elements at once.

**Note:** If nothing is returned in the setter function (ie. `function(index, attr){}`), or if `undefined` is returned, the current value is not changed. This is useful for selectively setting values only when certain criteria are met.

## Examples:

**Example:** Set some attributes for all <img>s in the page.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     img { padding:10px; }
6     div { color:red; font-size:24px; }
7   </style>
8   <script src="http://code.jquery.com/jquery-
9 </head>
10 <body>
11
12   <img />
13   <img />
14   <img />
15
16   <div><B>Attribute of Ajax</B></div>
17
18   <script>
19     $("img").attr({
20       src: "/resources/hat.gif",
21       title: "jQuery",
22       alt: "jQuery Logo"
23     });
24     $("div").text($("img").attr("alt"));
25   </script>
26
27 </body>
28 </html>
```

## Demo

**Example** Set the id for divs based on the position in the page.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     div { color:blue; }
6     span { color:red; }
7     b { font-weight:bolder; }
8   </style>
9   <script src="http://code.jquery.com/jquery-...
10 </head>
11 <body>
12
13   <div>Zero-th <span></span></div>
14   <div>First <span></span></div>
15   <div>Second <span></span></div>
16
17 <script>
18   $("div").attr("id", function (arr) {
19     return "div-id" + arr;
20   })
21   .each(function () {
22     $("span", this).html("(ID = '<b>' + this.id
23   });
24 </script>
25
26 </body>
27 </html>
```

## Demo

**Example** Set the src attribute from title attribute on the image.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-1.11.0.min.js"></script>
5 </head>
6 <body>
7
8 <img title="hat.gif" />
9
10 <script>
11 $("img").attr("src", function() {
12   return "/resources/" + this.title;
13 });
14 </script>
15
16 </body>
17 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# Attribute Contains Prefix Selector [name|="value"]

Categories: [Selectors](#) > [Attribute](#)

## attributeContainsPrefix selector

**Description:** *Selects elements that have the specified attribute with a value either equal to a given string or starting with that string followed by a hyphen (-).*

**jQuery( "[attribute|='value']" )**

version added: [1.0](#)

**attribute:** An attribute name.

**value:** An attribute value. Can be either an unquoted single word or a quoted string.

This selector was introduced into the CSS specification to handle language attributes.

## Example:

*Find all links with an hreflang attribute that is english.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5   a { display: inline-block; }
6   </style>
7   <script src="http://code.jquery.com/jquery-1.10.2.min.js"></script>
8 </head>
9 <body>
10
11   <a href="example.html" hreflang="en">Some text</a>
12
13   <a href="example.html" hreflang="en-UK">Some text</a>
14
15   <a href="example.html" hreflang="english">Some text</a>
16
17 <script>
18   $('a[hreflang="en"]').css('border', '3px dotted red');
19 </script>
20
21 </body>
22 </html>
```

Demo

**A new version of this book is available!**

---

# Attribute Contains Selector

## [name\*="value"]

Categories: [Selectors](#) > [Attribute](#)

## attributeContains selector

**Description:** *Selects elements that have the specified attribute with a value containing the a given substring.*

**jQuery( "[attribute\*='value']" )**

version added: [1.0](#)

**attribute:** An attribute name.

**value:** An attribute value. Can be either an unquoted single word or a quoted string.

This is the most generous of the jQuery attribute selectors that match against a value. It will select an element if the selector's string appears anywhere within the element's attribute value. Compare this selector with the Attribute Contains Word selector (e.g. `[attr~="word"]`), which is more appropriate in many cases.

## Example:

*Finds all inputs with a name attribute that contains 'man' and sets the value with some text.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-1.10.2.min.js"></script>
5 </head>
6 <body>
7   <input name="man-news" />
8
9   <input name="milkman" />
10  <input name="letterman2" />
11  <input name="newmilk" />
12  <script>$('input[name*="man"]').val('has man')</script>
13
14 </body>
15 </html>
```

Demo

POWERED BY HERONOTE

A new version of this book is available!

---

# Attribute Contains Word Selector `[name~="value"]`

Categories: [Selectors](#) > [Attribute](#)

## attributeContainsWord selector

**Description:** *Selects elements that have the specified attribute with a value containing a given word, delimited by spaces.*

**jQuery( "[attribute~='value']" )**

version added: [1.0](#)

**attribute:** An attribute name.

**value:** An attribute value. Can be either an unquoted single word or a quoted string.

This selector matches the test string against each word in the attribute value, where a "word" is defined as a string delimited by whitespace. The selector matches if the test string is exactly equal to any of the words.

## Example:

*Finds all inputs with a name attribute that contains the word 'man' and sets the value with some text.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-...
5 </head>
6 <body>
7   <input name="man-news" />
8
9   <input name="milk man" />
10  <input name="letterman2" />
11  <input name="newmilk" />
12  <script>$('input[name~="man"]').val('mr. man .
13
14 </body>
15 </html>
```

Demo

POWERED BY HERONOTE

A new version of this book is available!

---

# Attribute Ends With Selector [name\$="value"]

Categories: [Selectors](#) > [Attribute](#)

## attributeEndsWith selector

**Description:** *Selects elements that have the specified attribute with a value ending exactly with a given string. The comparison is case sensitive.*

**jQuery( "[attribute\$='value']" )**

version added: [1.0](#)

**attribute:** An attribute name.

**value:** An attribute value. Can be either an unquoted single word or a quoted string.

## Example:

*Finds all inputs with an attribute name that ends with 'letter' and puts text in them.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <script src="http://code.jquery.com/jquery-1.10.2.min.js"></script>
5  </head>
6  <body>
7    <input name="newsletter" />
8
9    <input name="milkman" />
10   <input name="jobletter" />
11   <script>$('input[name$="letter"]').val('a letter');
12
13 </body>
14 </html>
```

Demo

POWERED BY HERONOTE

A new version of this book is available!

---

# Attribute Equals Selector [name="value"]

Categories: [Selectors](#) > [Attribute](#)

## attributeEquals selector

**Description:** *Selects elements that have the specified attribute with a value exactly equal to a certain value.*

**jQuery( "[attribute='value']" )**

version added: [1.0](#)

**attribute:** An attribute name.

**value:** An attribute value. Can be either an unquoted single word or a quoted string.

## Example:

*Finds all inputs with a value of "Hot Fuzz" and changes the text of the next sibling span.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-1.10.2.min.js"></script>
5 </head>
6 <body>
7   <div>
8     <label>
9       <input type="radio" name="newsletter" value="name" />
10      <span>name?</span>
11    </label>
12  </div>
13  <div>
14    <label>
15      <input type="radio" name="newsletter" value="value" />
16      <span>value?</span>
17    </label>
18  </div>
19  <div>
20    <label>
21      <input type="radio" name="newsletter" value="value" />
22      <span>value?</span>
23    </label>
24  </div>
25  <script>$('input[value="Hot Fuzz"]').next().text("Hot Fuzz");
26 </script>
27 </body>
28 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# Attribute Not Equal Selector [name!="value"]

Categories: [Selectors](#) > [Attribute](#) | [Selectors](#) > [jQuery Extensions](#)

## attributeNotEqual selector

**Description:** *Select elements that either don't have the specified attribute, or do have the specified attribute but not with a certain value.*

**jQuery( "[attribute!='value']" )**

version added: [1.0](#)

**attribute:** An attribute name.

**value:** An attribute value. Can be either an unquoted single word or a quoted string.

This selector is equivalent to `:not([attr="value"])`.

### Additional Notes:

Because `[name!="value"]` is a jQuery extension and not part of the CSS specification, queries using `[name!="value"]` cannot take advantage of the performance boost provided by the native DOM `querySelectorAll()` method. For better performance in modern browsers, use `$("your-pure-css-selector").not('[name="value"]')` instead.

## Example:

*Finds all inputs that don't have the name 'newsletter' and appends text to the span next to it.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-1.10.2.min.js"></script>
5 </head>
6 <body>
7   <div>
8
9     <input type="radio" name="newsletter" value="Cold Fusion" />
10    <span>name is newsletter</span>
11
12  </div>
13  <div>
14    <input type="radio" value="Cold Fusion" />
15    <span>no name</span>
16
17  </div>
18  <div>
19    <input type="radio" name="accept" value="Cold Fusion" />
20
21    <span>name is accept</span>
22  </div>
23  <script>$('input[name!="newsletter"]').next().append("<br>");
24
25 </body>
26 </html>
```

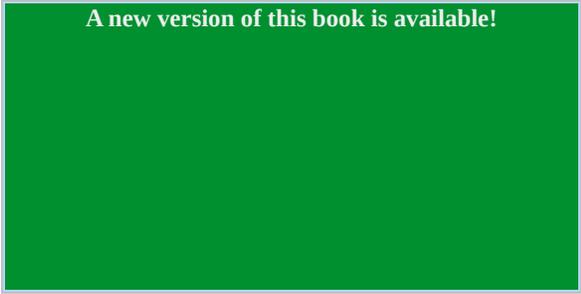


Demo

---

POWERED BY HERONOTE

A new version of this book is available!



---

# Attribute Starts With Selector [name^="value"]

Categories: [Selectors](#) > [Attribute](#)

## attributeStartsWith selector

**Description:** *Selects elements that have the specified attribute with a value beginning exactly with a given string.*

**jQuery( "[attribute^='value']" )**

version added: [1.0](#)

**attribute:** An attribute name.

**value:** An attribute value. Can be either an unquoted single word or a quoted string.

This selector can be useful for identifying elements in pages produced by server-side frameworks that produce HTML with systematic element IDs. However it will be slower than using a class selector so leverage classes, if you can, to group like elements.

## Example:

*Finds all inputs with an attribute name that starts with 'news' and puts text in them.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <script src="http://code.jquery.com/jquery-
5  </head>
6  <body>
7    <input name="newsletter" />
8
9    <input name="milkman" />
10   <input name="newsboy" />
11   <script>$('input[name^="news"]').val('news he
12
13 </body>
14 </html>
```

Demo

POWERED BY HERONOTE

A new version of this book is available!

---

# .before()

Categories: [Manipulation](#) > [DOM Insertion, Outside](#)

`.before( content [, content ] )`

Returns: [jQuery](#)

**Description:** *Insert content, specified by the parameter, before each element in the set of matched elements.*

**.before( content [, content ] )**

version added: [1.0](#)

**content**

Type: [htmlString](#) or [Element](#) or [jQuery](#)

HTML string, DOM element, or jQuery object to insert before each element in the set of matched elements.

**content**

Type: [htmlString](#) or [Element](#) or [Array](#) or [jQuery](#)

One or more additional DOM elements, arrays of elements, HTML strings, or jQuery objects to insert before each element in the set of matched elements.

**.before( function )**

version added: [1.4](#)

**function**

Type: [Function\(\)](#)

A function that returns an HTML string, DOM element(s), or jQuery object to insert before each element in the set of matched elements. Receives the index position of the element in the set as an argument. Within the function, `this` refers to the current element in the set.

The `.before()` and `.insertBefore()` methods perform the same task. The major difference is in the syntax—specifically, in the placement of the content and target. With `.before()`, the selector expression preceding the method is the container before which the content is inserted. With `.insertBefore()`, on the other hand, the content precedes the method, either as a selector expression or as markup created on the fly, and it is inserted before the target container.

Consider the following HTML:

```
1 | <div class="container">
```

```
2 | <h2>Greetings</h2>
3 | <div class="inner">Hello</div>
4 | <div class="inner">Goodbye</div>
5 | </div>
```

You can create content and insert it before several elements at once:

```
1 | $(' .inner ').before('<p>Test</p>');
```

Each inner `<div>` element gets this new content:

```
1 | <div class="container">
2 |   <h2>Greetings</h2>
3 |   <p>Test</p>
4 |   <div class="inner">Hello</div>
5 |   <p>Test</p>
6 |   <div class="inner">Goodbye</div>
7 | </div>
```

You can also select an element on the page and insert it before another:

```
1 | $(' .container ').before($(' h2 '));
```

If an element selected this way is inserted into a single location elsewhere in the DOM, it will be moved before the target (not cloned):

```
1 | <h2>Greetings</h2>
2 | <div class="container">
3 |   <div class="inner">Hello</div>
4 |   <div class="inner">Goodbye</div>
```

```
5 | </div>
```

If there is more than one target element, however, cloned copies of the inserted element will be created for each target after the first.

In jQuery 1.4, `.before()` and `.after()` will also work on disconnected DOM nodes:

```
1 | $("<div/>").before("<p></p>");
```

The result is a jQuery set that contains a paragraph and a div (in that order).

### Additional Arguments

Similar to other content-adding methods such as `.prepend()` and `.after()`, `.before()` also supports passing in multiple arguments as input. Supported input includes DOM elements, jQuery objects, HTML strings, and arrays of DOM elements.

For example, the following will insert two new `<div>`s and an existing `<div>` before the first paragraph:

```
1 | var $newdiv1 = $('<div id="object1"/>'),
2 |     newdiv2 = document.createElement('div'),
3 |     existingdiv1 = document.getElementById('f
4 |
5 | $('p').first().before($newdiv1, [newdiv2, exist
```

Since `.before()` can accept any number of additional arguments, the same result can be achieved by passing in the three `<div>`s as three separate arguments, like so: `$('p').first().before($newdiv1, newdiv2, existingdiv1)`. The type and number of arguments will largely depend on how you collect the elements in your code.



## Examples:

**Example:** *Inserts some HTML before all paragraphs.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>p { background:yellow; }</style>
5   <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
6 </head>
7 <body>
8   <p> is what I said...</p>
9   <script>$("p").before("<b>Hello</b>");</script>
10
11 </body>
12 </html>
```

Demo

**Example:** *Inserts a DOM element before all paragraphs.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>p { background:yellow; }</style>
5   <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
6 </head>
7 <body>
8   <p> is what I said...</p>
9   <script>$("p").before( document.createTextNode("<b>Hello</b>"));</script>
10
11 </body>
12 </html>
```

## Demo

**Example** Inserts a jQuery object (similar to an Array of DOM Elements) before all paragraphs.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>p { background:yellow; }</style>
5   <script src="http://code.jquery.com/jquery-
6 </head>
7 <body>
8   <p> is what I said...</p><b>Hello</b>
9   <script>$("p").before( $("b") );</script>
10
11 </body>
12 </html>
```

## Demo

POWERED BYHERONOTE

A new version of this book is available!

---

# .bind()

Categories: [Events](#) > [Event Handler Attachment](#)

`.bind( eventType [, eventData ], handler(eventObject) )`

Returns: [jQuery](#)

**Description:** *Attach a handler to an event for the elements.*

**.bind( eventType [, eventData ],  
handler(eventObject) )**

version added: [1.0](#)

**eventType**

Type: [String](#)

A string containing one or more DOM event types, such as "click" or "submit," or custom event names.

**eventData**

Type: [Object](#)

An object containing data that will be passed to the event handler.

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute each time the event is triggered.

**.bind( eventType [, eventData ],  
preventBubble )**

version added: [1.4.3](#)

**eventType**

Type: [String](#)

A string containing one or more DOM event types, such as "click" or "submit," or custom event names.

**eventData**

Type: [Object](#)

An object containing data that will be passed to the event handler.

**preventBubble**

Type: [Boolean](#)

Setting the third argument to false will attach a function that prevents the default action from occurring and stops the event

from bubbling. The default is true.

## `.bind( events )`

version added: [1.4](#)

### events

Type: [Object](#)

An object containing one or more DOM event types and functions to execute for them.

As of jQuery 1.7, the `.on()` method is the preferred method for attaching event handlers to a document. For earlier versions, the `.bind()` method is used for attaching an event handler directly to elements. Handlers are attached to the currently selected elements in the jQuery object, so those elements *must exist* at the point the call to `.bind()` occurs. For more flexible event binding, see the discussion of event delegation in `.on()` or `.delegate()`.

Any string is legal for `eventType`; if the string is not the name of a native DOM event, then the handler is bound to a custom event. These events are never called by the browser, but may be triggered manually from other JavaScript code using `.trigger()` or `.triggerHandler()`.

If the `eventType` string contains a period ( `.` ) character, then the event is namespaced. The period character separates the event from its namespace. For example, in the call `.bind('click.name', handler)`, the string `click` is the event type, and the string `name` is the namespace. Namespacing allows us to unbind or trigger some events of a type without affecting others. See the discussion of `.unbind()` for more information.

There are shorthand methods for some standard browser events such as `.click()` that can be used to attach or trigger event handlers. For a complete list of shorthand methods, see the [events category](#).

When an event reaches an element, all handlers bound to that event type for the element are fired. If there are multiple handlers registered, they will always execute in the order in which they were bound. After all handlers have executed, the event continues along the normal event propagation path.

A basic usage of `.bind()` is:

```
1 | $('#foo').bind('click', function() {  
2 |     alert('User clicked on "foo."');  
3 | });
```

This code will cause the element with an ID of `foo` to respond to the `click` event. When a user clicks inside this element thereafter, the alert will be shown.

## Multiple Events

Multiple event types can be bound at once by including each one separated by a space:

```
1 | $('#foo').bind('mouseenter mouseleave', funct:  
2 |     $(this).toggleClass('entered');  
3 | });
```

The effect of this on `<div id="foo">` (when it does not initially have the "entered" class) is to add the "entered" class when the mouse enters the `<div>` and remove the class when the mouse leaves.

As of jQuery 1.4 we can bind multiple event handlers simultaneously by passing an object of event type/handler pairs:

```
1 | $('#foo').bind({  
2 |     click: function() {  
3 |         // do something on click  
4 |     },  
5 |     mouseenter: function() {  
6 |         // do something on mouseenter  
7 |     }  
8 | });
```

## Event Handlers

The `handler` parameter takes a callback function, as shown above. Within the handler, the keyword `this` refers to the DOM element to which the handler is bound. To make use of the element in jQuery, it can be passed to the normal `$( )` function. For example:

```
1 | $('#foo').bind('click', function() {  
2 |     alert($(this).text());  
3 | });
```

After this code is executed, when the user clicks inside the element with an ID of `foo`, its text contents will be shown as an alert.

As of jQuery 1.4.2 duplicate event handlers can be bound to an element instead of being discarded. This is useful when the event data feature is being used, or when other unique data resides in a closure around the event handler function.

In jQuery 1.4.3 you can now pass in `false` in place of an event handler. This will bind an event handler equivalent to: `function(){ return false; }`. This function can be removed at a later time by calling: `.unbind( eventName, false )`.

## The Event object

The `handler` callback function can also take parameters. When the function is called, the event object will be passed to the first parameter.

The event object is often unnecessary and the parameter omitted, as sufficient context is usually available when the handler is bound to know exactly what needs to be done when the handler is triggered. However, at times it becomes necessary to gather more information about the user's environment at the time the event was initiated. [View the full Event Object.](#)

Returning `false` from a handler is equivalent to calling both `.preventDefault()` and `.stopPropagation()` on the event object.

Using the event object in a handler looks like this:

```
1 | $(document).ready(function() {
2 |     $('#foo').bind('click', function(event) {
3 |         alert('The mouse cursor is at ('
4 |             + event.pageX + ', ' + event.pageY + ')
5 |     });
6 | });
```

Note the parameter added to the anonymous function. This code will cause a click on the element with ID `foo` to report the page coordinates of the mouse cursor at the time of the click.

### Passing Event Data

The optional `eventData` parameter is not commonly used. When provided, this argument allows us to pass additional information to the handler. One handy use of this parameter is to work around issues caused by closures. For example, suppose we have two event handlers that both refer to the same external variable:

```
1 | var message = 'Spoon!';
2 | $('#foo').bind('click', function() {
3 |     alert(message);
4 | });
5 | message = 'Not in the face!';
6 | $('#bar').bind('click', function() {
7 |     alert(message);
8 | });
```

Because the handlers are closures that both have `message` in their environment, both will display the message `Not in the face!` when triggered. The variable's value has changed. To sidestep this, we can pass the message in using `eventData`:

```
1 | var message = 'Spoon!';
```

```
2 | $('#foo').bind('click', {msg: message}, function
3 |     alert(event.data.msg);
4 | });
5 | message = 'Not in the face!';
6 | $('#bar').bind('click', {msg: message}, function
7 |     alert(event.data.msg);
8 | });
```

This time the variable is not referred to directly within the handlers; instead, the variable is passed in *by value* through `eventData`, which fixes the value at the time the event is bound. The first handler will now display Spoon! while the second will alert Not in the face!

*Note that objects are passed to functions by reference, which further complicates this scenario.*

If `eventData` is present, it is the second argument to the `.bind()` method; if no additional data needs to be sent to the handler, then the callback is passed as the second and final argument.

*See the `.trigger()` method reference for a way to pass data to a handler at the time the event happens rather than when the handler is bound.*

As of jQuery 1.4 we can no longer attach data (and thus, events) to object, embed, or applet elements because critical errors occur when attaching data to Java applets.

**Note:** Although demonstrated in the next example, it is inadvisable to bind handlers to both the `click` and `dblclick` events for the same

element. The sequence of events triggered varies from browser to browser, with some receiving two click events before the `dblclick` and others only one. Double-click sensitivity (maximum time between clicks that is detected as a double click) can vary by operating system and browser, and is often user-configurable.

## Examples:

**Example:** Handle click and double-click for the paragraph. Note: the coordinates are window relative, so in this case relative to the demo iframe.

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5    p { background:yellow; font-weight:bold; cursor:
6    padding:5px; }
7    p.over { background: #ccc; }
8    span { color:red; }
9  </style>
10   <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
11 </head>
12 <body>
13   <p>Click or double click here.</p>
14   <span></span>
15   <script>
16     $("p").bind("click", function(event){
17     var str = "(" + event.pageX + ", " + event.pageY + ")";
18     $("span").text("Click happened! " + str);
19     });
20     $("p").bind("dblclick", function(){
21     $("span").text("Double-click happened in " + event.pageX + ", " + event.pageY);
22     });
23     $("p").bind("mouseenter mouseleave", function(event){
24     $(this).toggleClass("over");
25     });
26
27   </script>
28
29 </body>
```

30 | </html>

## Demo

**Example:** To display each paragraph's text in an alert box whenever it is clicked:

```
1 | $("p").bind("click", function(){
2 | alert( $(this).text() );
3 | });
```

**Example:** You can pass some extra data before the event handler:

```
1 | function handler(event) {
2 | alert(event.data.foo);
3 | }
4 | $("p").bind("click", {foo: "bar"}, handler)
```

**Example:** Cancel a default action and prevent it from bubbling up by returning `false`:

```
1 | $("form").bind("submit", function() { return false; });
```

**Example:** Cancel only the default action by using the `.preventDefault()` method.

```
1 | $("form").bind("submit", function(event) {
2 | event.preventDefault();
3 | });
```

**Example:** Stop an event from bubbling without preventing the default action by using the `.stopPropagation()` method.

```
1 | $("form").bind("submit", function(event) {  
2 |     event.stopPropagation();  
3 | });
```

**Example:** Bind custom events.

```
1 | <!DOCTYPE html>  
2 | <html>  
3 | <head>  
4 |     <style>  
5 | p { color:red; }  
6 | span { color:blue; }  
7 | </style>  
8 |     <script src="http://code.jquery.com/jquery-  
9 | </head>  
10 | <body>  
11 |     <p>Has an attached custom event.</p>  
12 | <button>Trigger custom event</button>  
13 | <span style="display:none;"></span>  
14 | <script>  
15 |  
16 | $( "p" ).bind( "myCustomEvent", function( e, myName ) {  
17 |     $( this ).text( myName + ", hi there!" );  
18 |     $( "span" ).stop().css( "opacity", 1 )  
19 |     .text( "myName = " + myName )  
20 |     .fadeIn( 30 ).fadeOut( 1000 );  
21 | });  
22 | $( "button" ).click( function () {  
23 |     $( "p" ).trigger( "myCustomEvent", [ "John" ] );  
24 | });  
25 |
```

```
26 </script>
27
28 </body>
29 </html>
```

Demo

**Example:** *Bind multiple events simultaneously.*

```
1 $( "div.test" ).bind({
2   click: function(){
3     $(this).addClass("active");
4   },
5   mouseenter: function(){
6     $(this).addClass("inside");
7   },
8   mouseleave: function(){
9     $(this).removeClass("inside");
10  }
11 });
```

POWERED BY **HERONOTE**

A new version of this book is available!

---

# .blur()

Categories: [Events](#) > [Form Events](#) | [Forms](#)

`.blur( handler(eventObject) )`

Returns: [jQuery](#)

**Description:** *Bind an event handler to the "blur" JavaScript event, or trigger that event on an element.*

**.blur( handler(eventObject) )**

version added: [1.0](#)

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute each time the event is triggered.

**.blur( [eventData ],  
handler(eventObject) )**

version added: [1.4.3](#)

**eventData**

Type: [Object](#)

An object containing data that will be passed to the event handler.

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute each time the event is triggered.

**.blur()**

version added: [1.0](#)

This method does not accept any arguments.

This method is a shortcut for `.on('blur', handler)` in the first two variations, and `.trigger('blur')` in the third.

The `blur` event is sent to an element when it loses focus. Originally, this event was only applicable to form elements, such as `<input>`. In recent browsers, the domain of the event has been extended to include all element types. An element can lose focus via keyboard commands, such as the Tab key, or by mouse clicks elsewhere on the page.

For example, consider the HTML:

```
1 | <form>
2 |   <input id="target" type="text" value="Field 1"
3 |   <input type="text" value="Field 2" />
4 | </form>
5 | <div id="other">
6 |   Trigger the handler
7 | </div>
8 | The event handler can be bound to the first input field
9 | $('#target').blur(function() {
10 |   alert('Handler for .blur() called.');
```

Now if the first field has the focus, clicking elsewhere or tabbing away from it displays the alert:

Handler for .blur() called.

To trigger the event programmatically, apply `.blur()` without an argument:

```
1 | $('#other').click(function() {
2 |   $('#target').blur();
3 | });
```

After this code executes, clicks on Trigger the handler will also alert the message.

The `blur` event does not bubble in Internet Explorer. Therefore, scripts that rely on event delegation with the `blur` event will not work consistently across browsers. As of version 1.4.2, however, jQuery works around this limitation by mapping `blur` to the `focusout` event in its event delegation methods, `.live()` and `.delegate()`.

## Example:

*To trigger the blur event on all paragraphs:*

```
1 | $("p").blur();
```

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# :button Selector

Categories: [Selectors](#) > [Form](#) | [Selectors](#) > [jQuery Extensions](#)

## button selector

**Description:** *Selects all button elements and elements of type button.*

**jQuery( ":button" )**

version added: [1.0](#)

An equivalent selector to `$(":button")` using valid CSS is `$("button, input[type='button']")`.

### Additional Notes:

Because `:button` is a jQuery extension and not part of the CSS specification, queries using `:button` cannot take advantage of the performance boost provided by the native DOM `querySelectorAll()` method. To achieve the best performance when using `:button` to select elements, first select the elements using a pure CSS selector, then use `.filter(":button")`.

## Example:

*Find all button inputs and mark them.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5      textarea { height:35px; }
6      div { color: red; }
7      fieldset { margin: 0; padding: 0; border-wid
8      .marked { background-color: yellow; border:
9    </style>
10   <script src="http://code.jquery.com/jquery-
11 </head>
12 <body>
13   <form>
14     <fieldset>
15       <input type="button" value="Input Button"
16       <input type="checkbox" />
17
18       <input type="file" />
19       <input type="hidden" />
20       <input type="image" />
21
22       <input type="password" />
23       <input type="radio" />
24       <input type="reset" />
25
26       <input type="submit" />
27       <input type="text" />
28       <select><option>Option<option/></select>
29
30       <textarea></textarea>
31       <button>Button</button>
```

```
32     </fieldset>
33 </form>
34
35 <div>
36 </div>
37 <script>
38     var input = $(":button").addClass("marked");
39     $("div").text( "For this type jQuery found " + input.length);
40     $("form").submit(function () { return false; });
41 </script>
42
43 </body>
44 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# callbacks.add()

Categories: [Callbacks Object](#)

`callbacks.add( callbacks )`

Returns: [Callbacks](#)

**Description:** *Add a callback or a collection of callbacks to a callback list.*

**callbacks.add( callbacks )**

version added: [1.7](#)

**callbacks**

Type: [Function\(\)](#) or [Array](#)

A function, or array of functions, that are to be added to the callback list.

This method returns the `Callbacks` object onto which it is attached (`this`).

## Example

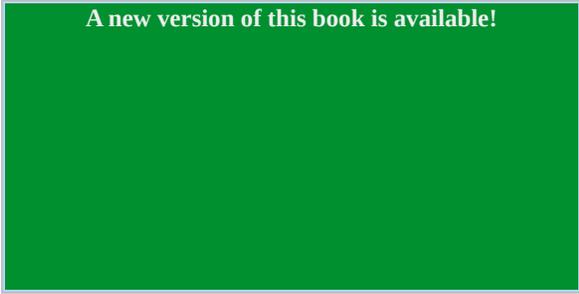
Use `callbacks.add()` to add new callbacks to a callback list:

```
1 // a sample logging function to be added to a
2 var foo = function( value ) {
3     console.log( "foo: " + value );
4 };
5
6 // another function to also be added to the
7 var bar = function( value ){
8     console.log( "bar: " + value );
9 };
10
11 var callbacks = $.Callbacks();
12
13 // add the function "foo" to the list
14 callbacks.add( foo );
15
16 // fire the items on the list
17 callbacks.fire( "hello" );
18 // outputs: "foo: hello"
19
20 // add the function "bar" to the list
21 callbacks.add( bar );
22
23 // fire the items on the list again
24 callbacks.fire( "world" );
25
26 // outputs:
27 // "foo: world"
28 // "bar: world"
```

---

POWERED BY **HERONOTE**

A new version of this book is available!



---

# callbacks.disable()

Categories: [Callbacks Object](#)

`callbacks.disable()`

Returns: [Callbacks](#)

**Description:** *Disable a callback list from doing anything more.*

`callbacks.disable()`

version added: [1.7](#)

This method does not accept any arguments.

This method returns the `Callbacks` object onto which it is attached (`this`).

## Example

Use `callbacks.disable()` to disable further calls to a callback list:

```
1 // a sample logging function to be added to a
2 var foo = function( value ) {
3     console.log( value );
4 };
5
6 var callbacks = $.Callbacks();
7
8 // add the above function to the list
9 callbacks.add( foo );
10
11 // fire the items on the list
12 callbacks.fire( "foo" );
13 // outputs: foo
14
15 // disable further calls being possible
16 callbacks.disable();
17
18 // attempt to fire with "foobar" as an argument
19 callbacks.fire( "foobar" );
20 // foobar isn't output
```

**A new version of this book is available!**

---

# callbacks.disabled()

Categories: [Callbacks Object](#)

`callbacks.disabled()`

Returns: [Boolean](#)

**Description:** *Determine if the callbacks list has been disabled.*

`callbacks.disabled()`

version added: [1.7](#)

This method does not accept any arguments.

## Example:

Use `callbacks.disabled()` to determine if the callbacks list has been disabled:

```
1 // a sample logging function to be added to a
2 var foo = function( value ) {
3     console.log( "foo:" + value );
4 };
5
6 var callbacks = $.Callbacks();
7
8 // add the logging function to the callback
9 callbacks.add( foo );
10
11 // fire the items on the list, passing an a
12 callbacks.fire( "hello" );
13 // outputs "foo: hello"
14
15 // disable the callbacks list
16 callbacks.disable();
17
18 // test the disabled state of the list
19 console.log ( callbacks.disabled() );
20 // outputs: true
```

**A new version of this book is available!**

---

# callbacks.empty()

Categories: [Callbacks Object](#)

`callbacks.empty()`

Returns: [Callbacks](#)

**Description:** *Remove all of the callbacks from a list.*

`callbacks.empty()`

version added: [1.7](#)

This method does not accept any arguments.

This method returns the Callbacks object onto which it is attached (`this`).

## Example

Use `callbacks.empty()` to empty a list of callbacks:

```
1 // a sample logging function to be added to a
2 var foo = function( value1, value2 ) {
3     console.log( "foo: " + value1 + ", " + value2 );
4 }
5
6 // another function to also be added to the list
7 var bar = function( value1, value2 ){
8     console.log( "bar: " + value1 + ", " + value2 );
9 }
10
11 var callbacks = $.Callbacks();
12
13 // add the two functions
14 callbacks.add( foo );
15 callbacks.add( bar );
16
17 // empty the callbacks list
18 callbacks.empty();
19
20 // check to ensure all callbacks have been removed
21 console.log( callbacks.has( foo ) );
22 // false
23 console.log( callbacks.has( bar ) );
24 // false
```

**A new version of this book is available!**

---

# callbacks.fire()

Categories: [Callbacks Object](#)

`callbacks.fire( arguments )`

Returns: [Callbacks](#)

**Description:** *Call all of the callbacks with the given arguments*

**callbacks.fire( arguments )**

version added: [1.7](#)

**arguments**

Type: [Anything](#)

The argument or list of arguments to pass back to the callback list.

This method returns the Callbacks object onto which it is attached ([this](#)).

## Example

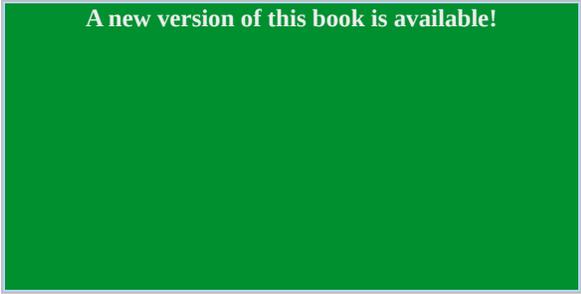
Use `callbacks.fire()` to invoke the callbacks in a list with any arguments that have been passed:

```
1 // a sample logging function to be added to a
2 var foo = function( value ) {
3     console.log( "foo:" + value );
4 }
5
6 var callbacks = $.Callbacks();
7
8 // add the function "foo" to the list
9 callbacks.add( foo );
10
11 // fire the items on the list
12 callbacks.fire( "hello" ); // outputs: "foo:
13 callbacks.fire( "world" ); // outputs: "foo:
14
15 // add another function to the list
16 var bar = function( value ){
17     console.log( "bar:" + value );
18 }
19
20 // add this function to the list
21 callbacks.add( bar );
22
23 // fire the items on the list again
24 callbacks.fire( "hello again" );
25 // outputs:
26 // "foo: hello again"
27 // "bar: hello again"
```

---

POWERED BY HERONOTE

A new version of this book is available!



---

# callbacks.fired()

Categories: [Callbacks Object](#)

`callbacks.fired()`

Returns: [Boolean](#)

**Description:** *Determine if the callbacks have already been called at least once.*

`callbacks.fired()`

version added: [1.7](#)

This method does not accept any arguments.

## Example

Use `callbacks.fired()` to determine if the callbacks in a list have been called at least once:

```
1 // a sample logging function to be added to a
2 var foo = function( value ) {
3   console.log( "foo:" + value );
4 };
5
6 var callbacks = $.Callbacks();
7
8 // add the function "foo" to the list
9 callbacks.add( foo );
10
11 // fire the items on the list
12 callbacks.fire( "hello" ); // outputs: "foo:
13 callbacks.fire( "world" ); // outputs: "foo:
14
15 // test to establish if the callbacks have be
16 console.log( callbacks.fired() );
```

POWERED BY HERONOTE

A new version of this book is available!

---

# callbacks.fireWith()

Categories: [Callbacks Object](#)

`callbacks.fireWith( [context ] [, args ] )` Returns: [Callbacks](#)

**Description:** *Call all callbacks in a list with the given context and arguments.*

`callbacks.fireWith( [context ] [, args ]` version added: [1.7](#)  
`)`

**context**

Type:

A reference to the context in which the callbacks in the list should be fired.

**args**

Type:

An argument, or array of arguments, to pass to the callbacks in the list.

This method returns the `Callbacks` object onto which it is attached (`this`).

## Example

Use `callbacks.fireWith()` to fire a list of callbacks with a specific context and an array of arguments:

```
1 // a sample logging function to be added to a
2 var log = function( value1, value2 ) {
3     console.log( "Received: " + value1 + ", " +
4     };
5
6 var callbacks = $.Callbacks();
7
8 // add the log method to the callbacks list
9 callbacks.add( log );
10
11 // fire the callbacks on the list using the c
12 // and an arguments array
13
14 callbacks.fireWith( window, ["foo", "bar"]);
15
16 // outputs: "Received: foo, bar"
```

POWERED BY HERONOTE

A new version of this book is available!

---

# callbacks.has()

Categories: [Callbacks Object](#)

`callbacks.has( callback )`

Returns: [Boolean](#)

**Description:** *Determine whether a supplied callback is in a list*

**callbacks.has( callback )**

version added: [1.7](#)

**callback**

Type: [Function\(\)](#)

The callback to search for.

## Example:

Use `callbacks.has()` to check if a callback list contains a specific callback:

```
1 // a sample logging function to be added to a
2 var foo = function( value1, value2 ) {
3   console.log( "Received: " + value1 + ", " +
4 };
5
6 // a second function which will not be added
7 var bar = function( value1, value2 ) {
8   console.log( "foobar" );
9 }
10
11 var callbacks = $.Callbacks();
12
13 // add the log method to the callbacks list
14 callbacks.add( foo );
15
16 // determine which callbacks are in the list
17
18 console.log( callbacks.has( foo ) );
19 // true
20 console.log( callbacks.has( bar ) );
21 // false
```

**A new version of this book is available!**

---

# callbacks.lock()

Categories: [Callbacks Object](#)

`callbacks.lock()`

Returns: [Callbacks](#)

**Description:** *Lock a callback list in its current state.*

`callbacks.lock()`

version added: [1.7](#)

This method does not accept any arguments.

This method returns the Callbacks object onto which it is attached (`this`).

If the Callbacks object is created with the `"memory"` flag as its argument, additional functions may be added and fired after the callback list is locked.

## Examples:

**Example:** Use `callbacks.lock()` to lock a callback list to avoid further changes being made to the list state:

```
1 // a sample logging function to be added to a
2 var foo = function( value ) {
3   console.log( "foo:" + value );
4 };
5
6 var callbacks = $.Callbacks();
7
8 // add the logging function to the callback l
9 callbacks.add( foo );
10
11 // fire the items on the list, passing an arg
12 callbacks.fire( "hello" );
13 // outputs "foo: hello"
14
15 // lock the callbacks list
16 callbacks.lock();
17
18 // try firing the items again
19 callbacks.fire( "world" );
20
21 // as the list was locked, no items
22 // were called, so "world" isn't logged
```

**Example:** Use `callbacks.lock()` to lock a callback list with "memory," and then resume using the list:

```
1 <!DOCTYPE html>
2 <html>
```

```
3 <head>
4   <script src="http://code.jquery.com/jquery-latest.js">
5 </head>
6 <body>
7   <div id="log"></div>
8 <script>// simple function for logging results
9 var log = function( value ) {
10   $( "#log" ).append( "<p>" + value + "</p>" );
11 };
12
13 // two sample functions to be added to a callbacks list
14 var foo = function( value ) {
15   log( "foo: " + value );
16 };
17 var bar = function( value ) {
18   log( "bar: " + value );
19 };
20
21 // create the callbacks object with the "memory" flag
22 var callbacks = $.Callbacks( "memory" );
23
24 // add the foo logging function to the callback list
25 callbacks.add( foo );
26
27 // fire the items on the list, passing an argument
28 callbacks.fire( "hello" );
29 // outputs "foo: hello"
30
31 // lock the callbacks list
32 callbacks.lock();
33
34 // try firing the items again
35 callbacks.fire( "world" );
36 // as the list was locked, no items were called,
37 // so "foo: world" isn't logged
38
39 // add the foo function to the callback list again
```

```
40 callbacks.add( foo );
41
42 // try firing the items again
43 callbacks.fire( "silentArgument" );
44 // outputs "foo: hello" because the argument value was .
45
46 // add the bar function to the callback list
47 callbacks.add( bar );
48
49 callbacks.fire( "youHadMeAtHello" );
50 // outputs "bar: hello" because the list is still locked
51 // and the argument value is still stored in memory
52 </script>
53
54 </body>
55 </html>
```

Demo

POWERED BY HERONOTE

A new version of this book is available!

---

# callbacks.locked()

Categories: [Callbacks Object](#)

`callbacks.locked()`

Returns: [Boolean](#)

**Description:** *Determine if the callbacks list has been locked.*

`callbacks.locked()`

version added: [1.7](#)

This method does not accept any arguments.

## Example:

Use `callbacks.locked()` to determine the lock-state of a callback list:

```
1 // a sample logging function to be added to a
2 var foo = function( value ) {
3   console.log( "foo: " + value);
4 };
5
6 var callbacks = $.Callbacks();
7
8 // add the logging function to the callback l.
9 callbacks.add( foo );
10
11 // fire the items on the list, passing an arg
12 callbacks.fire( "hello" );
13 // outputs "foo: hello"
14
15 // lock the callbacks list
16 callbacks.lock();
17
18 // test the lock-state of the list
19 console.log ( callbacks.locked() );
20 // true
```

**A new version of this book is available!**

---

# callbacks.remove()

Categories: [Callbacks Object](#)

`callbacks.remove( callbacks )`

Returns: [Callbacks](#)

**Description:** *Remove a callback or a collection of callbacks from a callback list.*

**callbacks.remove( callbacks )**

version added: [1.7](#)

**callbacks**

Type: [Function\(\)](#) or [Array](#)

A function, or array of functions, that are to be removed from the callback list.

This method returns the Callbacks object onto which it is attached (`this`).

## Example:

Use `callbacks.remove()` to remove callbacks from a callback list:

```
1 // a sample logging function to be added to a
2 var foo = function( value ) {
3     console.log( "foo: " + value );
4 };
5
6 var callbacks = $.Callbacks();
7
8 // add the function "foo" to the list
9 callbacks.add( foo );
10
11 // fire the items on the list
12 callbacks.fire( "hello" );
13 // outputs: "foo: hello"
14
15 // remove "foo" from the callback list
16 callbacks.remove( foo );
17
18 // fire the items on the list again
19 callbacks.fire( "world" );
20
21 // nothing output as "foo" is no longer in the
```

**A new version of this book is available!**

---

# .change()

Categories: [Events](#) > [Form Events](#) | [Forms](#)

`.change( handler(eventObject) )`

Returns: [jQuery](#)

**Description:** *Bind an event handler to the "change" JavaScript event, or trigger that event on an element.*

`.change( handler(eventObject) )`

version added: [1.0](#)

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute each time the event is triggered.

`.change( [eventData ],  
handler(eventObject) )`

version added: [1.4.3](#)

**eventData**

Type: [Object](#)

An object containing data that will be passed to the event handler.

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute each time the event is triggered.

`.change()`

version added: [1.0](#)

This method does not accept any arguments.

This method is a shortcut for `.on('change', handler)` in the first two variations, and `.trigger('change')` in the third.

The `change` event is sent to an element when its value changes. This event is limited to `<input>` elements, `<textarea>` boxes and `<select>` elements. For select boxes, checkboxes, and radio buttons, the event is fired immediately when the user makes a selection with the mouse, but for the other element types the event is deferred until the element loses focus.

For example, consider the HTML:

```
1 | <form>
2 |   <input class="target" type="text" value="Field 1"
3 |   <select class="target">
4 |     <option value="option1" selected="selected"
5 |     <option value="option2">Option 2</option>
6 |   </select>
7 | </form>
8 | <div id="other">
9 |   Trigger the handler
10| </div>
```

The event handler can be bound to the text input and the select box:

```
1 | $(' .target' ).change(function() {
2 |   alert('Handler for .change() called. ');
3 | });
```

Now when the second option is selected from the dropdown, the alert is displayed. It is also displayed if you change the text in the field and then click away. If the field loses focus without the contents having changed, though, the event is not triggered. To trigger the event manually, apply `.change()` without arguments:

```
1 | $('#other' ).click(function() {
2 |   $(' .target' ).change();
3 | });
```

After this code executes, clicks on Trigger the handler will also alert the message. The message will display twice, because the handler has been bound to the `change` event on both of the form elements.

As of jQuery 1.4, the `change` event bubbles in Internet Explorer, behaving consistently with the event in other modern browsers.



## Examples:

**Example:** *Attaches a change event to the select that gets the text for each selected option and writes them in the div. It then triggers the event for the initial text draw.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5      div { color:red; }
6    </style>
7    <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
8  </head>
9  <body>
10   <select name="sweets" multiple="multiple">
11     <option>Chocolate</option>
12     <option selected="selected">Candy</option>
13     <option>Taffy</option>
14     <option selected="selected">Caramel</option>
15     <option>Fudge</option>
16     <option>Cookie</option>
17   </select>
18   <div></div>
19 </body>
20 </html>
21 <script>
22 $( "select" ).change( function () {
23   var str = "";
24   $( "select option:selected" ).each( function ( i, option ) {
25     str += $( this ).text() + " ";
26   });
27   $( "div" ).text( str );
28 })
```

```
29 | .change();
30 | </script>
31 |
32 | </body>
33 | </html>
```

## Demo

**Example:** *To add a validity test to all text input elements:*

```
1 | $("input[type='text']").change( function() {
2 |     // check input ($(this).val()) for validity
3 | });
```

POWERED BY HERONOTE

A new version of this book is available!

---

# :checkbox Selector

Categories: [Selectors](#) > [Form](#) | [Selectors](#) > [jQuery Extensions](#)

## checkbox selector

**Description:** *Selects all elements of type checkbox.*

**jQuery( ":checkbox" )**

version added: [1.0](#)

`$( ':checkbox' )` is equivalent to `$( '[type=checkbox]' )`. As with other pseudo-class selectors (those that begin with a ":" ) it is recommended to precede it with a tag name or some other selector; otherwise, the universal selector ("\*") is implied. In other words, the bare `$( ':checkbox' )` is equivalent to `$( '*:checkbox' )`, so `$( 'input:checkbox' )` should be used instead.

### Additional Notes:

Because `:checkbox` is a jQuery extension and not part of the CSS specification, queries using `:checkbox` cannot take advantage of the performance boost provided by the native DOM `querySelectorAll()` method. For better performance in modern browsers, use `[type="checkbox"]` instead.

## Example:

*Finds all checkbox inputs.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     textarea { height:25px; }
6   </style>
7   <script src="http://code.jquery.com/jquery-
8 </head>
9 <body>
10  <form>
11    <input type="button" value="Input Button"
12    <input type="checkbox" />
13
14    <input type="checkbox" />
15    <input type="file" />
16    <input type="hidden" />
17
18    <input type="image" />
19    <input type="password" />
20    <input type="radio" />
21
22    <input type="reset" />
23    <input type="submit" />
24    <input type="text" />
25
26    <select><option>Option</option></select>
27    <textarea></textarea>
28    <button>Button</button>
29  </form>
30
31  <div>
```

```
32     </div>
33     <script>
34
35         var input = $("form input:checkbox").wrap('<span></
36         $("div").text("For this type jQuery found "
37             .css("color", "red");
38         $("form").submit(function () { return false
39
40     </script>
41
42 </body>
43 </html>
```

Demo

POWERED BYHERONOTE

A new version of this book is available!

---

# :checked Selector

Categories: [Selectors](#) > [Form](#)

## checked selector

**Description:** *Matches all elements that are checked.*

**jQuery( ":checked" )**

version added: [1.0](#)

The `:checked` selector works for checkboxes and radio buttons. For select elements, use the `:selected` selector.

## Examples:

**Example:** Determine how many input elements are checked.

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5      div { color:red; }
6    </style>
7    <script src="http://code.jquery.com/jquery-...
8  </head>
9  <body>
10
11  <form>
12    <p>
13      <input type="checkbox" name="newsletter" >
14
15      <input type="checkbox" name="newsletter" >
16      <input type="checkbox" name="newsletter" >
17
18      <input type="checkbox" name="newsletter" >
19      <input type="checkbox" name="newsletter" >
20    </p>
21  </form>
22  <div></div>
23
24  <script>
25  var countChecked = function() {
26    var n = $( "input:checked" ).length;
27    $( "div" ).text( n + ( n === 1 ? " is" : " a
28  };
29  countChecked();
30
```

```
31 $( "input[type=checkbox]" ).on( "click", countChecked )
32 </script>
33
34 </body>
35 </html>
```

Demo

**Example:** *Identify the checked radio input.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     input, label { line-height: 1.5em; }
6   </style>
7   <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
8 </head>
9 <body>
10
11 <form>
12   <div>
13     <input type="radio" name="fruit" value="orange" checked="" />
14     <label for="orange">orange</label>
15   </div>
16   <div>
17     <input type="radio" name="fruit" value="apple" />
18     <label for="apple">apple</label>
19   </div>
20   <div>
21     <input type="radio" name="fruit" value="banana" />
22     <label for="banana">banana</label>
23   </div>
24   <div id="log"></div>
25 </form>
26
```

```
27 <script>
28 $( "input" ).on( "click", function() {
29     $( "#log" ).html( $("input:checked").val() +
30     });
31 </script>
32
33 </body>
34 </html>
```

Demo

POWERED BY HERONOTE

A new version of this book is available!

---

# Child Selector (“parent > child”)

Categories: [Selectors](#) > [Hierarchy](#)

## child selector

**Description:** *Selects all direct child elements specified by "child" of elements specified by "parent".*

**jQuery( "parent > child" )**

version added: [1.0](#)

**parent:** Any valid selector.

**child:** A selector to filter the child elements.

As a CSS selector, the child combinator is supported by all modern web browsers including Safari, Firefox, Opera, Chrome, and Internet Explorer 7 and above, but notably not by Internet Explorer versions 6 and below. However, in jQuery, this selector (along with all others) works across all supported browsers, including IE6.

The child combinator (E > F) can be thought of as a more specific form of the descendant combinator (E F) in that it selects only first-level descendants.

## Example:

*Places a border around all list items that are children of `<ul class="topnav">` .*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5  body { font-size:14px; }
6  </style>
7    <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
8  </head>
9  <body>
10
11  <ul class="topnav">
12    <li>Item 1</li>
13    <li>Item 2
14      <ul><li>Nested item 1</li><li>Nested item 2</li></ul>
15    </li>
16    <li>Item 3</li>
17  </ul>
18
19  <script>$("ul.topnav > li").css("border", "3px solid red");</script>
20
21  </body>
22  </html>
```

Demo

**A new version of this book is available!**

---

# .children()

Categories: [Traversing](#) > [Tree Traversal](#)

`.children( [selector ] )`

Returns: [jQuery](#)

**Description:** *Get the children of each element in the set of matched elements, optionally filtered by a selector.*

`.children( [selector ] )`

version added: [1.0](#)

**selector**

Type: [Selector](#)

A string containing a selector expression to match elements against.

Given a jQuery object that represents a set of DOM elements, the `.children()` method allows us to search through the children of these elements in the DOM tree and construct a new jQuery object from the matching elements. The `.children()` method differs from `.find()` in that `.children()` only travels a single level down the DOM tree while `.find()` can traverse down multiple levels to select descendant elements (grandchildren, etc.) as well. Note also that like most jQuery methods, `.children()` does not return text nodes; to get *all* children including text and comment nodes, use `.contents()`.

The `.children()` method optionally accepts a selector expression of the same type that we can pass to the `$( )` function. If the selector is supplied, the elements will be filtered by testing whether they match it.

Consider a page with a basic nested list on it:

```
1 <ul class="level-1">
2   <li class="item-i">I</li>
3   <li class="item-ii">II
4     <ul class="level-2">
5       <li class="item-a">A</li>
6       <li class="item-b">B
7         <ul class="level-3">
8           <li class="item-1">1</li>
```

```
9         <li class="item-2">2</li>
10        <li class="item-3">3</li>
11    </ul>
12 </li>
13 <li class="item-c">C</li>
14 </ul>
15 </li>
16 <li class="item-iii">III</li>
17 </ul>
```

If we begin at the level-2 list, we can find its children:

```
1 | $('ul.level-2').children().css('background-co:
```

The result of this call is a red background behind items A, B, and C. Since we do not supply a selector expression, all of the children are part of the returned jQuery object. If we had supplied one, only the matching items among these three would be included.

## Examples:

**Example:** *Find all children of the clicked element.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     body { font-size:16px; font-weight:bolder;
6     div { width:130px; height:82px; margin:10px
7         border:1px solid blue; padding:4px; }
8     #container { width:auto; height:105px; marg
9         border:none; }
10    .hilite { border-color:red; }
11    #results { display:block; color:red; }
12    p { margin:10px; border:1px solid transparen
13    span { color:blue; border:1px solid transpa
14    input { width:100px; }
15    em { border:1px solid transparent; }
16    a { border:1px solid transparent; }
17    b { border:1px solid transparent; }
18    button { border:1px solid transparent; }
19  </style>
20  <script src="http://code.jquery.com/jquery-
21 </head>
22 <body>
23   <div id="container">
24
25     <div>
26       <p>This <span>is the <em>way</em> we</s
27       write <em>the</em> demo,</p>
28
29     </div>
30     <div>
31       <a href="#"><b>w</b>rit<b>e</b></a> the
```

```

32     the</button> demo,
33 </div>
34
35 <div>
36     This <span>the way we <em>write</em> the
37
38     <input type="text" value="early" /> in
39 </div>
40 <p>
41     <span>t</span>he <span>m</span>orning.
42     <span id="results">Found <span>0</span> children :
43
44 </p>
45 </div>
46 <script>
47
48     $("#container").click(function (e) {
49         $("*").removeClass("hilite");
50         var $kids = $(e.target).children();
51         var len = $kids.addClass("hilite").length;
52
53         $("#results span:first").text(len);
54         $("#results span:last").text(e.target.tagName);
55
56         e.preventDefault();
57         return false;
58     });
59 </script>
60
61 </body>
62 </html>

```

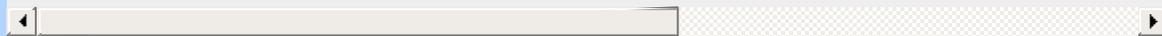
Demo

**Example:** *Find all children of each div.*

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5    body { font-size:16px; font-weight:bolder; }
6    span { color:blue; }
7    p { margin:5px 0; }
8    </style>
9    <script src="http://code.jquery.com/jquery-latest.js">
10 </head>
11 <body>
12   <p>Hello (this is a paragraph)</p>
13
14   <div><span>Hello Again (this span is a child of the a
15   <p>And <span>Again</span> (in another paragraph)
16
17   <div>And One Last <span>Time</span> (most text direct.
18 <script>$("div").children().css("border-bottom"
19
20 </body>
21 </html>

```



Demo

**Example:** Find all children with a class "selected" of each div.

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5
6    body { font-size:16px; font-weight:bolder;
7    p { margin:5px 0; }
8    </style>
9    <script src="http://code.jquery.com/jquery-

```

```
10 </head>
11 <body>
12   <div>
13     <span>Hello</span>
14     <p class="selected">Hello Again</p>
15     <div class="selected">And Again</div>
16
17     <p>And One Last Time</p>
18   </div>
19   <script>$("div").children(".selected").css("color"
20
21 </body>
22 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# Class Selector (“`.class`”)

Categories: [Selectors](#) > [Basic](#)

## class selector

**Description:** *Selects all elements with the given class.*

**jQuery( ".class" )**

version added: [1.0](#)

**class:** A class to search for. An element can have multiple classes; only one of them must match.

For class selectors, jQuery uses JavaScript's native `getElementsByClassName()` function if the browser supports it.

## Examples:

**Example:** Finds the element with the class "myClass".

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     div, span {
6       width: 100px;
7       height: 40px;
8       float:left;
9       padding: 10px;
10      margin: 10px;
11      background-color: #EEEEEE;
12    }
13  </style>
14  <script src="http://code.jquery.com/jquery-
15 </head>
16 <body>
17   <div class="notMe">div class="notMe"</div>
18
19   <div class="myClass">div class="myClass"</d.
20   <span class="myClass">span class="myClass"<.
21 <script>$(".myClass").css("border", "3px solid
22
23 </body>
24 </html>
```

Demo

**Example:** Finds the element with both "myclass" and "otherclass" classes.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     div,span {
6       width: 100px;
7       height: 40px;
8       float:left;
9       padding: 10px;
10      margin: 10px;
11      background-color: #EEEEEE;
12    }
13  </style>
14  <script src="http://code.jquery.com/jquery-latest.js">
15 </head>
16 <body>
17   <div class="myclass">div class="notMe"</div>
18
19   <div class="myclass otherclass">div class="myClass"
20   <span class="myclass otherclass">span class="myClass"
21 <script>$(".myclass.otherclass").css("border", "13px solid
22
23 </body>
24 </html>
```

Demo

POWERED BY HERONOTE

A new version of this book is available!

---

# .clearQueue()

Categories: [Effects](#) > [Custom](#) | [Data](#) | [Utilities](#)

`.clearQueue( [queueName ] )`

Returns: [jQuery](#)

**Description:** *Remove from the queue all items that have not yet been run.*

`.clearQueue( [queueName ] )`

version added: [1.4](#)

**queueName**

Type: [String](#)

A string containing the name of the queue. Defaults to `fx`, the standard effects queue.

When the `.clearQueue()` method is called, all functions on the queue that have not been executed are removed from the queue. When used without an argument, `.clearQueue()` removes the remaining functions from `fx`, the standard effects queue. In this way it is similar to `.stop(true)`. However, while the `.stop()` method is meant to be used only with animations, `.clearQueue()` can also be used to remove any function that has been added to a generic jQuery queue with the `.queue()` method.

## Example:

*Empty the queue.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5    div { margin:3px; width:40px; height:40px;
6          position:absolute; left:0px; top:30px;
7          background:green; display:none; }
8    div.newcolor { background:blue; }
9    </style>
10   <script src="http://code.jquery.com/jquery-
11 </head>
12 <body>
13   <button id="start">Start</button>
14   <button id="stop">Stop</button>
15   <div></div>
16   <script>
17   $( "#start" ).click( function () {
18
19     var myDiv = $( "div" );
20     myDiv.show( "slow" );
21     myDiv.animate( { left: '+=200' }, 5000 );
22     myDiv.queue( function () {
23       var _this = $( this );
24       _this.addClass( "newcolor" );
25       _this.dequeue();
26     });
27
28     myDiv.animate( { left: '-=200' }, 1500 );
29     myDiv.queue( function () {
30       var _this = $( this );
31       _this.removeClass( "newcolor" );
```

```
32     _this.dequeue();
33     });
34     myDiv.slideUp();
35
36 });
37
38 $("#stop").click(function () {
39     var myDiv = $("#div");
40     myDiv.clearQueue();
41     myDiv.stop();
42 });</script>
43
44 </body>
45 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# .click()

Categories: [Events](#) > [Mouse Events](#)

`.click( handler(eventObject) )`

Returns: [jQuery](#)

**Description:** *Bind an event handler to the "click" JavaScript event, or trigger that event on an element.*

`.click( handler(eventObject) )`

version added: [1.0](#)

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute each time the event is triggered.

`.click( [eventData ],  
handler(eventObject) )`

version added: [1.4.3](#)

**eventData**

Type: [Object](#)

An object containing data that will be passed to the event handler.

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute each time the event is triggered.

`.click()`

version added: [1.0](#)

This method does not accept any arguments.

This method is a shortcut for `.on('click', handler)` in the first two variations, and `.trigger('click')` in the third. The `click` event is sent to an element when the mouse pointer is over the element, and the mouse button is pressed and released. Any HTML element can receive this event. For example, consider the HTML:

```
1 | <div id="target">  
2 |   Click here  
3 | </div>  
4 | <div id="other">  
5 |   Trigger the handler
```

6 | `</div>`

Click here

Trigger the handler

The event handler can be bound to any `<div>`:

```
1 | $( "#target" ).click( function() {  
2 |     alert( "Handler for .click() called." );  
3 | });
```

Now if we click on this element, the alert is displayed:

Handler for .click() called.

We can also trigger the event when a different element is clicked:

```
1 | $( "#other" ).click( function() {  
2 |     $( "#target" ).click();  
3 | });
```

After this code executes, clicks on Trigger the handler will also alert the message.

The `click` event is only triggered after this exact series of events:

- The mouse button is depressed while the pointer is inside the element.

- The mouse button is released while the pointer is inside the element.

This is usually the desired sequence before taking an action. If this is not required, the `mousedown` or `mouseup` event may be more suitable.



## Examples:

**Example:** *Hide paragraphs on a page when they are clicked:*

```
1 | <!DOCTYPE html>
2 | <html>
3 | <head>
4 |   <style>
5 |     p { color:red; margin:5px; cursor:pointer;
6 |     p:hover { background:yellow; }
7 |   </style>
8 |   <script src="http://code.jquery.com/jquery-
9 | </head>
10 | <body>
11 |   <p>First Paragraph</p>
12 |
13 |   <p>Second Paragraph</p>
14 |   <p>Yet one more Paragraph</p>
15 | <script>
16 |   $("p").click(function () {
17 |     $(this).slideUp();
18 |   });
19 | </script>
20 |
21 | </body>
22 | </html>
```

Demo

**Example:** *Trigger the click event on all of the paragraphs on the page:*

```
1 | $("p").click();
```

POWERED BY HERONOTE

A new version of this book is available!

---

# .clone()

Categories: [Manipulation](#) > [Copying](#)

`.clone( [withDataAndEvents ] )`

Returns: [jQuery](#)

**Description:** *Create a deep copy of the set of matched elements.*

`.clone( [withDataAndEvents ] )`

version added: [1.0](#)

**withDataAndEvents** (default: `false`)

Type: [Boolean](#)

A Boolean indicating whether event handlers should be copied along with the elements. As of jQuery 1.4, element data will be copied as well.

`.clone( [withDataAndEvents ] [,  
deepWithDataAndEvents ] )`

version added: [1.5](#)

**withDataAndEvents** (default: `false`)

Type: [Boolean](#)

A Boolean indicating whether event handlers and data should be copied along with the elements. The default value is `false`.  
*\*In jQuery 1.5.0 the default value was incorrectly `true`; it was changed back to `false` in 1.5.1 and up.*

**deepWithDataAndEvents** (default: `value of  
withDataAndEvents`)

Type: [Boolean](#)

A Boolean indicating whether event handlers and data for all children of the cloned element should be copied. By default its value matches the first argument's value (which defaults to `false`).

The `.clone()` method performs a *deep* copy of the set of matched elements, meaning that it copies the matched elements as well as all of their descendant elements and text nodes. When used in conjunction with one of the insertion methods, `.clone()` is a convenient way to duplicate elements on a page. Consider the following HTML:

```
1 | <div class="container">
2 |   <div class="hello">Hello</div>
3 |   <div class="goodbye">Goodbye</div>
4 | </div>
```

As shown in the discussion for `$.append()`, normally when an element is inserted somewhere in the DOM, it is moved from its old location. So, given the code:

```
1 | $(' .hello' ).appendTo( ' .goodbye' );
```

The resulting DOM structure would be:

```
1 | <div class="container">
2 |   <div class="goodbye">
3 |     Goodbye
4 |     <div class="hello">Hello</div>
5 |   </div>
6 | </div>
```

To prevent this and instead create a copy of the element, you could write the following:

```
1 | $(' .hello' ).clone().appendTo( ' .goodbye' );
```

This would produce:

```
1 | <div class="container">
2 |   <div class="hello">Hello</div>
3 |   <div class="goodbye">
4 |     Goodbye
5 |     <div class="hello">Hello</div>
```

```
6 | </div>
7 | </div>
```

**Note:** When using the `.clone()` method, you can modify the cloned elements or their contents before (re-)inserting them into the document.

Normally, any event handlers bound to the original element are *not* copied to the clone. The optional `withDataAndEvents` parameter allows us to change this behavior, and to instead make copies of all of the event handlers as well, bound to the new copy of the element. As of jQuery 1.4, all element data (attached by the `.data()` method) is also copied to the new copy.

However, objects and arrays within element data are not copied and will continue to be shared between the cloned element and the original element. To deep copy all data, copy each one manually:

```
1 | var $elem = $('#elem').data( "arr": [ 1 ] ),
2 |     $clone = $elem.clone( true )
3 |     .data( "arr", $.extend( [], $elem.data("arr") ) );
```

As of jQuery 1.5, `withDataAndEvents` can be optionally enhanced with `deepWithDataAndEvents` to copy the events and data for all children of the cloned element.

**Note:** Using `.clone()` has the side-effect of producing elements with duplicate `id` attributes, which are supposed to be unique. Where possible, it is recommended

*to avoid cloning elements with this attribute  
or using `class` attributes as identifiers  
instead.*

## Examples:

**Example:** Clones all **h** elements (and selects the clones) and prepends them to all paragraphs.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-
5 </head>
6 <body>
7
8   <b>Hello</b><p>, how are you?</p>
9
10  <script>
11    $("b").clone().prependTo("p");
12  </script>
13
14 </body>
15 </html>
```

### Demo

**Example:** When using `clone()` to clone a collection of elements that are not attached to the DOM their order when inserted into the DOM is not guaranteed. However it may be possible to preserve sort order with a workaround, as demonstrated:

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     #orig, #copy, #copy-correct {
```

```
6     float: left;
7     width: 20%;
8 }
9 </style>
10 <script src="http://code.jquery.com/jquery-latest.js">
11 </head>
12 <body>
13
14 <div id="orig">
15     <div class="elem"><a>1</a></div>
16     <div class="elem"><a>2</a></div>
17     <div class="elem"><a>3</a></div>
18     <div class="elem"><a>4</a></div>
19     <div class="elem"><a>5</a></div>
20 </div>
21 <div id="copy"></div>
22 <div id="copy-correct"></div>
23
24 <script>
25 // sort order is not guaranteed here and may vary with
26 $('#copy').append($('#orig .elem')
27     .clone()
28     .children('a')
29     .prepend('foo - ')
30     .parent()
31     .clone());
32
33 // correct way to approach where order is maintained
34 $('#copy-correct')
35     .append($('#orig .elem')
36     .clone()
37     .children('a')
38     .prepend('bar - ')
39     .end());
40 </script>
41
42 </body>
```

43 | </html>

Demo

POWERED BYHERONOTE

A new version of this book is available!

---

# .closest()

Categories: [Traversing](#) > [Tree Traversal](#)

## Contents:

[.closest\( selector \)](#)

`.closest( selector )`

`.closest( selector [, context ] )`

`.closest( jQuery object )`

`.closest( element )`

[.closest\( selectors \[, context \] \)](#)

`.closest( selectors [, context ] )`

`.closest( selector )`

Returns: [jQuery](#)

**Description:** For each element in the set, get the first element that matches the selector by testing the element itself and traversing up through its ancestors in the DOM tree.

**.closest( selector )**

version added: [1.3](#)

**selector**

Type: [Selector](#)

A string containing a selector expression to match elements against.

**.closest( selector [, context ] )**

version added: [1.4](#)

**selector**

Type: [Selector](#)

A string containing a selector expression to match elements against.

**context**

Type: [Element](#)

A DOM element within which a matching element may be found. If no context is passed in then the context of the jQuery set will be used instead.

**.closest( jQuery object )**

version added: [1.6](#)

**jQuery object**

Type: [jQuery](#)

A jQuery object to match elements against.

**.closest( element )**

version added: [1.6](#)

**element**

Type: [Element](#)

An element to match elements against.

Given a jQuery object that represents a set of DOM elements, the

`.closest()` method searches through these elements and their ancestors in the DOM tree and constructs a new jQuery object from the matching elements. The `.parents()` and `.closest()` methods are similar in that they both traverse up the DOM tree. The differences between the two, though subtle, are significant:

| <code>.closest()</code>                                                                      | <code>.parents()</code>                                                                                                                                                                |
|----------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Begins with the current element                                                              | Begins with the parent element                                                                                                                                                         |
| Travels up the DOM tree until it finds a match for the supplied selector                     | Travels up the DOM tree to the document's root element, adding each ancestor element to a temporary collection; it then filters that collection based on a selector if one is supplied |
| The returned jQuery object contains zero or one element for each element in the original set | The returned jQuery object contains zero or more elements for each element in the original set                                                                                         |

```
1 <ul id="one" class="level-1">
2   <li class="item-i">I</li>
3   <li id="ii" class="item-ii">II
4     <ul class="level-2">
5       <li class="item-a">A</li>
6       <li class="item-b">B
7         <ul class="level-3">
8           <li class="item-1">1</li>
9           <li class="item-2">2</li>
10          <li class="item-3">3</li>
11        </ul>
12      </li>
13      <li class="item-c">C</li>
14    </ul>
15  </li>
16  <li class="item-iii">III</li>
17 </ul>
```

Suppose we perform a search for `<ul>` elements starting at item A:

```
1 | $( 'li.item-a' ).closest( 'ul' )
2 |   .css( 'background-color', 'red' );
```

This will change the color of the level-2 `<ul>`, since it is the first encountered when traveling up the DOM tree.

Suppose we search for an `<li>` element instead:

```
1 | $( 'li.item-a' ).closest( 'li' )
2 |   .css( 'background-color', 'red' );
```

This will change the color of list item A. The `.closest()` method begins its search *with the element itself* before progressing up the DOM tree, and stops when item A matches the selector.

We can pass in a DOM element as the context within which to search for the closest element.

```
1 | var listItemII = document.getElementById( 'ii' )
2 | $( 'li.item-a' ).closest( 'ul', listItemII )
3 |   .css( 'background-color', 'red' );
4 | $( 'li.item-a' ).closest( '#one', listItemII )
5 |   .css( 'background-color', 'green' );
```

This will change the color of the level-2 `<ul>`, because it is both the first `<ul>` ancestor of list item A and a descendant of list item II. It will not change the color of the level-1 `<ul>`, however, because it is not a descendant of list item II.

## Examples:

**Example:** Show how event delegation can be done with `closest`. The closest list element toggles a yellow background when it or its descendent is clicked.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     li { margin: 3px; padding: 3px; background:
6     li.highlight { background: yellow; }
7   </style>
8   <script src="http://code.jquery.com/jquery-
9 </head>
10 <body>
11   <ul>
12     <li><b>Click me!</b></li>
13     <li>You can also <b>Click me!</b></li>
14   </ul>
15 <script>
16   $( document ).on( "click", function( e ) {
17     $( e.target ).closest( "li" ).toggleClass( "
18   });
19 </script>
20
21 </body>
22 </html>
```

Demo

**Example:** Pass a jQuery object to `closest`. The closest list element toggles a yellow background when it or its descendent is clicked.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     li { margin: 3px; padding: 3px; background: #EEEEEE
6     li.hilight { background: yellow; }
7   </style>
8   <script src="http://code.jquery.com/jquery-latest.js"
9 </head>
10 <body>
11   <ul>
12     <li><b>Click me!</b></li>
13     <li>You can also <b>Click me!</b></li>
14   </ul>
15 <script>
16   var $listElements = $("li").css("color", "blue"
17   $( document ).on("click", function( e ) {
18     $( e.target ).closest( $listElements ).toggleClass(
19     });
20 </script>
21
22 </body>
23 </html>
```

Demo

`.closest( selectors [, context ] )` *Returns:* [Array](#)  
version deprecated: [1.7](#), removed: [1.8](#)

**Description:** *Get an array of all the elements and selectors matched against the current element up through the DOM tree.*

`.closest( selectors [, context ] )` version added: [1.4](#)

**selectors**

Type: [Array](#)

An array or string containing a selector expression to match elements against (can also be a jQuery object).

**context**

Type: [Element](#)

A DOM element within which a matching element may be found. If no context is passed in then the context of the jQuery set will be used instead.

**This signature (only!) is deprecated as of jQuery 1.7 and removed in jQuery 1.8. It was primarily meant to be used internally or by plugin authors.**

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# :contains() Selector

Categories: [Selectors](#) > [Content Filter](#)

## contains selector

**Description:** *Select all elements that contain the specified text.*

**jQuery( ":contains(text)" )**

version added: [1.1.4](#)

**text:** A string of text to look for. It's case sensitive.

The matching text can appear directly within the selected element, in any of that element's descendants, or a combination thereof. As with attribute value selectors, text inside the parentheses of `:contains()` can be written as a bare word or surrounded by quotation marks. The text must have matching case to be selected.

## Example:

*Find all divs containing "John" and underlines them.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-1.10.2.min.js"></script>
5 </head>
6 <body>
7
8 <div>John Resig</div>
9
10 <div>George Martin</div>
11 <div>Malcom John Sinclair</div>
12 <div>J. Ohn</div>
13
14 <script>
15   $("div:contains('John')").css("text-decoration: underline");
16 </script>
17 </body>
18 </html>
```

Demo

**A new version of this book is available!**

---

# .contents()

Categories: [Traversing](#) > [Miscellaneous Traversing](#)

`.contents()`

Returns: [jQuery](#)

**Description:** *Get the children of each element in the set of matched elements, including text and comment nodes.*

`.contents()`

version added: [1.2](#)

This method does not accept any arguments.

Given a jQuery object that represents a set of DOM elements, the `.contents()` method allows us to search through the immediate children of these elements in the DOM tree and construct a new jQuery object from the matching elements. The `.contents()` and `.children()` methods are similar, except that the former includes text nodes as well as HTML elements in the resulting jQuery object.

The `.contents()` method can also be used to get the content document of an iframe, if the iframe is on the same domain as the main page.

Consider a simple `<div>` with a number of text nodes, each of which is separated by two line break elements (`<br />`):

```
1 | <div class="container">
2 |   Lorem ipsum dolor sit amet, consectetur ad
3 |   do eiusmod tempor incididunt ut labore et c
4 |   <br /><br />
5 |   Ut enim ad minim veniam, quis nostrud exerc
6 |   laboris nisi ut aliquip ex ea commodo conse
7 |   <br /> <br />
8 |   Duis aute irure dolor in reprehenderit in v
9 |   esse cillum dolore eu fugiat nulla pariatur
10| </div>
```

We can employ the `.contents()` method to help convert this blob of text into three well-formed paragraphs:

```
1 | $( '.container' ).contents().filter(function()  
2 |     return this.nodeType == 3;  
3 | })  
4 |     .wrap( '<p></p>' )  
5 |     .end()  
6 |     .filter( 'br' )  
7 |     .remove();
```

This code first retrieves the contents of `<div class="container">` and then filters it for text nodes, which are wrapped in paragraph tags. This is accomplished by testing the `.nodeType` property of the element. This DOM property holds a numeric code indicating the node's type; text nodes use the code 3. The contents are again filtered, this time for `<br />` elements, and these elements are removed.

## Examples:

**Example:** Find all the text nodes inside a paragraph and wrap them with a bold tag.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-1.10.2.min.js"></script>
5 </head>
6 <body>
7   <p>Hello <a href="http://ejohn.org/">John</a>
8   <script>$("p").contents().filter(function(){
9     return this.nodeType !== 3;
10  }).each(function(){
11    $(this).wrap("<b>");
12  });
13 </script>
14 </body>
15 </html>
```

Demo

**Example:** Change the background colour of links inside of an iframe.

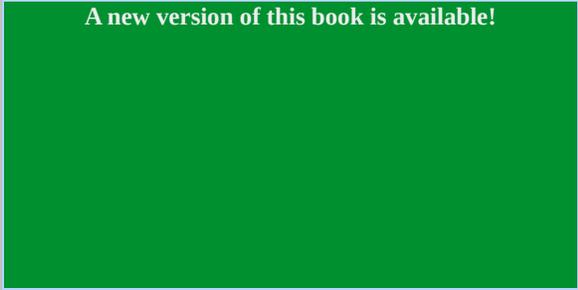
```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-1.10.2.min.js"></script>
5 </head>
6 <body>
7   <iframe src="http://api.jquery.com/" width="100%" height="100%"></iframe>
8   <script>$("#frameDemo").contents().find("a").css("background-color", "red");
9 </script>
10 </body>
11 </html>
```



Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

# .context

Categories: [Internals](#) | [Properties](#) > [Properties of jQuery Object Instances](#)

context

Returns: [Element](#)

**Description:** *The DOM node context originally passed to `jQuery()`; if none was passed then context will likely be the document.*

context

version added: [1.3](#)

The `.live()` method for binding event handlers uses this property to determine the root element to use for its event delegation needs.

The value of this property is typically equal to `document`, as this is the default context for jQuery objects if none is supplied. The context may differ if, for example, the object was created by searching within an `<iframe>` or XML document.

Note that the context property may only apply to the elements originally selected by `jQuery()`, as it is possible for the user to add elements to the collection via methods such as `.add()` and these may have a different context.

## Example:

*Determine the exact context used.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5    body { cursor:pointer; }
6    div { width:50px; height:30px; margin:5px;
7          background:green; }
8    span { color:red; }
9    </style>
10   <script src="http://code.jquery.com/jquery-
11 </head>
12 <body>
13   Context:<ul></ul>
14   <script>$("ul")
15     .append("<li>" + $("ul").context + "</li>")
16     .append("<li>" + $("ul", document.body).con
17
18   </script>
19
20 </body>
21 </html>
```

Demo

**A new version of this book is available!**

---

# .css()

Categories: [CSS](#) | [Manipulation](#) > [Style Properties](#)

Get the value of a style property for the first element in the set of matched elements or set one or more CSS properties for every matched element.

## Contents:

[.css\( propertyName \)](#)

.css( propertyName )

.css( propertyNamees )

[.css\( propertyName, value \)](#)

.css( propertyName, value )

.css( propertyName, function(index, value) )

.css( properties )

`.css( propertyName )`

Returns: [String](#)

**Description:** *Get the value of style properties for the first element in the set of matched elements.*

`.css( propertyName )`

version added: [1.0](#)

**propertyName**

Type: [String](#)

A CSS property.

`.css( propertyNames )`

version added: [1.9](#)

**propertyNames**

Type: [Array](#)

An array of one or more CSS properties.

The `.css()` method is a convenient way to get a style property from the first matched element, especially in light of the different ways browsers access most of those properties (the `getComputedStyle()` method in standards-based browsers versus the `currentStyle` and `runtimeStyle` properties in Internet Explorer) and the different terms browsers use for certain properties. For example, Internet Explorer's DOM implementation refers to the `float` property as `styleFloat`, while W3C standards-compliant browsers refer to it as `cssFloat`. For consistency, you can simply use `"float"`, and jQuery will translate it to the correct value for each browser.

Also, jQuery can equally interpret the CSS and DOM formatting of multiple-word properties. For example, jQuery understands and returns the correct value for both `.css('background-color')` and `.css('backgroundColor')`. Different browsers may return CSS color values that are logically but not textually equal, e.g., `#FFF`, `#ffffff`, and `rgb(255,255,255)`.

Shorthand CSS properties (e.g. `margin`, `background`, `border`) are not supported. For example, if you want to retrieve the rendered margin, use: `$(elem).css('marginTop')` and `$(elem).css('marginRight')`, and so on.

As of jQuery 1.9, passing an array of style properties to `.css()` will result in an object of property-value pairs.

## Examples:

**Example:** *Get the background color of a clicked div.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5    div { width:60px; height:60px; margin:5px; float:left; }
6    </style>
7    <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
8  </head>
9  <body>
10
11  <span id="result">&nbsp;   </span>
12  <div style="background-color:blue;"></div>
13  <div style="background-color:rgb(15,99,30);"></div>
14
15  <div style="background-color:#123456;"></div>
16  <div style="background-color:#f11;"></div>
17  <script>
18  $("div").click(function () {
19    var color = $(this).css("background-color");
20    $("#result").html("That div is <span style=" +
21      color + ";'>" + color +
22    });
23
24  </script>
25
26  </body>
27  </html>
```

Demo

**Example:** Get the width, height, text color, and background color of a clicked div.

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5  div { height: 50px; margin: 5px; padding: 5px;
6
7  #box1 { width: 50px; color: yellow; background-color: #f11; }
8  #box2 { width: 80px; color: rgb(255,255,255); background-color: #f11; }
9  #box3 { width: 40px; color: #fcc; background-color: #f11; }
10 #box4 { width: 70px; background-color: #f11; }
11 </style>
12 <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
13 </head>
14 <body>
15
16 <p id="result">&nbsp;</p>
17 <div id="box1">1</div>
18 <div id="box2">2</div>
19 <div id="box3">3</div>
20 <div id="box4">4</div>
21 <script>
22 $( "div" ).click( function () {
23   var html = [ "The clicked div has the following style properties: " ];
24
25   var styleProps = $( this ).css( [ "width", "height", "color", "background-color" ] );
26   $.each( styleProps, function( prop, value ) {
27     html.push( prop + ": " + value );
28   });
29
30   $( "#result" ).html( html.join( "<br>" ) );
31 });
32
33 </script>
```

```
34  
35 </body>  
36 </html>
```

Demo

`.css( propertyName, value )`

Returns: [jQuery](#)

**Description:** *Set one or more CSS properties for the set of matched elements.*

`.css( propertyName, value )`

version added: [1.0](#)

**propertyName**

Type: [String](#)

A CSS property name.

**value**

Type: [String](#) or [Number](#)

A value to set for the property.

`.css( propertyName, function(index, value) )`

version added: [1.4](#)

**propertyName**

Type: [String](#)

A CSS property name.

**function(index, value)**

Type: [Function\(\)](#)

A function returning the value to set. `this` is the current element. Receives the index position of the element in the set and the old value as arguments.

`.css( properties )`

version added: [1.0](#)

**properties**

Type: [PlainObject](#)

An object of property-value pairs to set.

As with the `.prop()` method, the `.css()` method makes setting properties of elements quick and easy. This method can take either a property name and value as separate parameters, or a single object of key-value pairs.

Also, jQuery can equally interpret the CSS and DOM formatting of

multiple-word properties. For example, jQuery understands and returns the correct value for both `.css({'background-color': '#ffe', 'border-left': '5px solid #ccc'})` and `.css({background-color: '#ffe', borderLeft: '5px solid #ccc'})`. Notice that with the DOM notation, quotation marks around the property names are optional, but with CSS notation they're required due to the hyphen in the name.

When using `.css()` as a setter, jQuery modifies the element's `style` property. For example, `$('#mydiv').css('color', 'green')` is equivalent to `document.getElementById('mydiv').style.color = 'green'`. Setting the value of a style property to an empty string — e.g. `$('#mydiv').css('color', '')` — removes that property from an element if it has already been directly applied, whether in the HTML style attribute, through jQuery's `.css()` method, or through direct DOM manipulation of the `style` property. It does not, however, remove a style that has been applied with a CSS rule in a stylesheet or `<style>` element. **Warning:** one notable exception is that, for IE 8 and below, removing a shorthand property such as `border` or `background` will remove that style entirely from the element, regardless of what is set in a stylesheet or `<style>` element.

As of jQuery 1.6, `.css()` accepts relative values similar to `.animate()`. Relative values are a string starting with `+=` or `-=` to increment or decrement the current value. For example, if an element's padding-left was 10px, `.css("padding-left", "+=15")` would result in a total padding-left of 25px.

As of jQuery 1.4, `.css()` allows us to pass a function as the property value:

```
1 | $('#div.example').css('width', function(index)
2 |     return index * 50;
3 | });
```

This example sets the widths of the matched elements to incrementally larger values.

**Note:** If nothing is returned in the setter function (ie. `function(index,`

`style){}`), or if `undefined` is returned, the current value is not changed. This is useful for selectively setting values only when certain criteria are met.

## Examples:

**Example:** To change the color of any paragraph to red on mouseover event.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     p { color:blue; width:200px; font-size:14px
6   </style>
7   <script src="http://code.jquery.com/jquery-
8 </head>
9 <body>
10
11   <p>Just roll the mouse over me.</p>
12
13   <p>Or me to see a color change.</p>
14
15 <script>
16   $( "p" ).mouseover( function ( ) {
17     $( this ).css( "color", "red" );
18   } );
19 </script>
20
21 </body>
22 </html>
```

Demo

**Example:** Increase the width of #box by 200 pixels

```
1 <!DOCTYPE html>
2 <html>
```

```

3 <head>
4   <style>
5     #box { background: black; color: snow; width:
6   </style>
7   <script src="http://code.jquery.com/jquery-latest.js"
8 </head>
9 <body>
10
11   <div id="box">Click me to grow</div>
12
13 <script>
14   $( "#box" ).one( "click", function () {
15     $( this ).css( "width", "+=200" );
16   });
17 </script>
18
19 </body>
20 </html>

```

Demo

**Example:** *To highlight a clicked word in the paragraph.*

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     p { color:blue; font-weight:bold; cursor:po.
6   </style>
7   <script src="http://code.jquery.com/jquery-.
8 </head>
9 <body>
10
11 <p>
12   Once upon a time there was a man
13   who lived in a pizza parlor. This

```

```

14 man just loved pizza and ate it all
15 the time. He went on to be the
16 happiest man in the world. The end.
17 </p>
18 <script>
19 var words = $("p:first").text().split(" ");
20 var text = words.join("</span> <span>");
21 $("p:first").html("<span>" + text + "</span>");
22 $("span").click(function () {
23     $(this).css("background-color", "yellow");
24 });
25
26 </script>
27
28 </body>
29 </html>

```

## Demo

**Example:** To set the color of all paragraphs to red and background to blue:

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     p { color:green; }
6   </style>
7   <script src="http://code.jquery.com/jquery-...
8 </head>
9 <body>
10
11   <p>Move the mouse over a paragraph.</p>
12   <p>Like this one or the one above.</p>
13
14 <script>

```

```

15  $("p").hover(function () {
16      $(this).css({'background-color' : 'yellow' ,
17  }, function () {
18      var cssObj = {
19          'background-color' : '#ddd',
20          'font-weight' : '',
21          'color' : 'rgb(0,40,244)')
22      };
23      $(this).css(cssObj);
24  });
25  </script>
26
27  </body>
28  </html>

```

Demo

**Example:** *Increase the size of a div when you click it:*

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4      <style>
5          div { width: 20px; height: 15px; background
6          </style>
7          <script src="http://code.jquery.com/jquery-
8  </head>
9  <body>
10
11     <div>click</div>
12     <div>click</div>
13
14  <script>
15      $("div").click(function() {
16          $(this).css({
17              width: function(index, value) {

```

```
18     return parseFloat(value) * 1.2;
19   },
20   height: function(index, value) {
21     return parseFloat(value) * 1.2;
22   }
23
24   });
25 });
26 </script>
27
28 </body>
29 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# .data()

Categories: [Data](#) | [Miscellaneous](#) > [Data Storage](#)

Store arbitrary data associated with the matched elements or return the value at the named data store for the first element in the set of matched elements.

## Contents:

[.data\( key, value \)](#)

[.data\( key, value \)](#)

[.data\( obj \)](#)

[.data\( key \)](#)

[.data\( key \)](#)

[.data\(\)](#)

`.data( key, value )`

Returns: [jQuery](#)

**Description:** *Store arbitrary data associated with the matched elements.*

`.data( key, value )`

version added: [1.2.3](#)

**key**

Type: [String](#)

A string naming the piece of data to set.

**value**

Type: [Object](#)

The new data value; it can be any Javascript type including Array or Object.

`.data( obj )`

version added: [1.4.3](#)

**obj**

Type: [Object](#)

An object of key-value pairs of data to update.

The `.data()` method allows us to attach data of any type to DOM elements in a way that is safe from circular references and therefore from memory leaks.

We can set several distinct values for a single element and retrieve them later:

```
1 | $( 'body' ).data( 'foo', 52 );  
2 | $( 'body' ).data( 'bar', { myType: 'test', count  
3 |  
4 | $( 'body' ).data( 'foo' ); // 52  
5 | $( 'body' ).data(); // {foo: 52, bar: { myType:
```

In jQuery 1.4.3 setting an element's data object with `.data(obj)`

extends the data previously stored with that element. jQuery itself uses the `.data()` method to save information under the names 'events' and 'handle', and also reserves any data name starting with an underscore ('\_') for internal use.

Prior to jQuery 1.4.3 (starting in jQuery 1.4) the `.data()` method completely replaced all data, instead of just extending the data object. If you are using third-party plugins it may not be advisable to completely replace the element's data object, since plugins may have also set data.

Due to the way browsers interact with plugins and external code, the `.data()` method cannot be used on `<object>` (unless it's a Flash plugin), `<applet>` or `<embed>` elements.

## Additional Notes:

Note that this method currently does not provide cross-platform support for setting data on XML documents, as Internet Explorer does not allow data to be attached via expando properties.

## Example:

*Store then retrieve a value from the div element.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5      div { color:blue; }
6      span { color:red; }
7    </style>
8    <script src="http://code.jquery.com/jquery-1.10.2.min.js"></script>
9  </head>
10 <body>
11   <div>
12     The values stored were
13     <span></span>
14     and
15     <span></span>
16   </div>
17   <script>
18     $("div").data("test", { first: 16, last: "pizza" });
19     $("span:first").text($("div").data("test").first);
20     $("span:last").text($("div").data("test").last);
21   </script>
22
23 </body>
24 </html>
```

Demo

`.data( key )`

Returns: [Object](#)

**Description:** Return the value at the named data store for the first element in the jQuery collection, as set by `data(name, value)` or by an HTML5 `data-*` attribute.

`.data( key )`

version added: [1.2.3](#)

**key**

Type: [String](#)

Name of the data stored.

`.data()`

version added: [1.4](#)

This method does not accept any arguments.

The `.data()` method allows us to attach data of any type to DOM elements in a way that is safe from circular references and therefore from memory leaks. We can retrieve several distinct values for a single element one at a time, or as a set:

```
1 | alert($('body').data('foo'));
2 | alert($('body').data());
```

The above lines alert the data values that were set on the `body` element. If no data at all was set on that element, `undefined` is returned.

```
1 | alert( $("body").data("foo")); //undefined
2 | $("body").data("bar", "foobar");
3 | alert( $("body").data("bar")); //foobar
```

## [HTML5 data-\\* Attributes](#)

As of jQuery 1.4.3 [HTML 5 data- attributes](#) will be automatically

pulled in to jQuery's data object. The treatment of attributes with embedded dashes was changed in jQuery 1.6 to conform to the [W3C HTML5 specification](#).

For example, given the following HTML:

```
1 | <div data-role="page" data-last-value="43" data-hidden="true" data-options="{name: 'John'}">
```

All of the following jQuery code will work.

```
1 | $("div").data("role") === "page";
2 | $("div").data("lastValue") === 43;
3 | $("div").data("hidden") === true;
4 | $("div").data("options").name === "John";
```

Every attempt is made to convert the string to a JavaScript value (this includes booleans, numbers, objects, arrays, and null) otherwise it is left as a string. To retrieve the value's attribute as a string without any attempt to convert it, use the `attr()` method. When the data attribute is an object (starts with '{') or array (starts with '[') then `jQuery.parseJSON` is used to parse the string; it must follow [valid JSON syntax](#) including quoted property names. The data- attributes are pulled in the first time the data property is accessed and then are no longer accessed or mutated (all data values are then stored internally in jQuery).

Calling `.data()` with no parameters retrieves all of the values as a JavaScript object. This object can be safely cached in a variable as long as a new object is not set with `.data(obj)`. Using the object directly to get or set values is faster than making individual calls to `.data()` to get or set each value:

```
1 | var mydata = $("#mydiv").data();
2 | if ( mydata.count < 9 ) {
3 |     mydata.count = 43;
```

```
4 | mydata.status = "embiggened";  
5 | }
```

## Additional Notes:

Note that this method currently does not provide cross-platform support for setting data on XML documents, as Internet Explorer does not allow data to be attached via expando properties.

## Example:

*Get the data named "blah" stored at for an element.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     div { margin:5px; background:yellow; }
6     button { margin:5px; font-size:14px; }
7     p { margin:5px; color:blue; }
8     span { color:red; }
9   </style>
10  <script src="http://code.jquery.com/jquery-
11 </head>
12 <body>
13   <div>A div</div>
14   <button>Get "blah" from the div</button>
15   <button>Set "blah" to "hello"</button>
16
17   <button>Set "blah" to 86</button>
18   <button>Remove "blah" from the div</button>
19   <p>The "blah" value of this div is <span>?</
20 <script>
21 $( "button" ).click( function( e ) {
22   var value;
23
24   switch ( $( "button" ).index( this ) ) {
25     case 0 :
26       value = $( "div" ).data( "blah" );
27       break;
28     case 1 :
29       $( "div" ).data( "blah", "hello" );
30       value = "Stored!";
31     break;
```

```
32     case 2 :
33         $("div").data("blah", 86);
34         value = "Stored!";
35         break;
36     case 3 :
37         $("div").removeData("blah");
38         value = "Removed!";
39         break;
40     }
41
42     $("span").text("" + value);
43 });
44
45 </script>
46
47 </body>
48 </html>
```

Demo

POWERED BYHERONOTE

A new version of this book is available!

---

# .dblclick()

Categories: [Events](#) > [Mouse Events](#)

`.dblclick( handler(eventObject) )`

Returns: [jQuery](#)

**Description:** *Bind an event handler to the "dblclick" JavaScript event, or trigger that event on an element.*

**.dblclick( handler(eventObject) )**

version added: [1.0](#)

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute each time the event is triggered.

**.dblclick( [eventData ],  
handler(eventObject) )**

version added: [1.4.3](#)

**eventData**

Type: [Object](#)

An object containing data that will be passed to the event handler.

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute each time the event is triggered.

**.dblclick()**

version added: [1.0](#)

This method does not accept any arguments.

This method is a shortcut for `.on('dblclick', handler)` in the first two variations, and `.trigger('dblclick')` in the third. The `dblclick` event is sent to an element when the element is double-clicked. Any HTML element can receive this event. For example, consider the HTML:

```
1 | <div id="target">
2 |   Double-click here
3 | </div>
4 | <div id="other">
5 |   Trigger the handler
```

6 | `</div>`

Double-click here

Trigger the handler

The event handler can be bound to any `<div>`:

```
1 | $( '#target' ).dblclick(function() {  
2 |     alert('Handler for .dblclick() called.');
```

```
3 | });
```

Now double-clicking on this element displays the alert:

Handler for `.dblclick()` called.

To trigger the event manually, apply `.dblclick()` without an argument:

```
1 | $( '#other' ).click(function() {  
2 |     $( '#target' ).dblclick();  
3 | });
```

After this code executes, (single) clicks on Trigger the handler will also alert the message.

The `dblclick` event is only triggered after this exact series of events:

- The mouse button is depressed while the pointer is inside the element.

- The mouse button is released while the pointer is inside the element.

- The mouse button is depressed again while the pointer is inside the element, within a time window that is system-dependent.

The mouse button is released while the pointer is inside the element.

It is inadvisable to bind handlers to both the `click` and `dblclick` events for the same element. The sequence of events triggered varies from browser to browser, with some receiving two `click` events before the `dblclick` and others only one. Double-click sensitivity (maximum time between clicks that is detected as a double click) can vary by operating system and browser, and is often user-configurable.

## Examples:

**Example:** To bind a "Hello World!" alert box the *dblclick* event on every paragraph on the page:

```
1 | $("p").dblclick( function () { alert("Hello Wo
```

**Example:** Double click to toggle background color.

```
1 | <!DOCTYPE html>
2 | <html>
3 | <head>
4 |   <style>
5 |     div { background:blue;
6 |           color:white;
7 |           height:100px;
8 |           width:150px;
9 |   }
10 |   div.dbl { background:yellow;color:black; }
11 | </style>
12 |   <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
13 | </head>
14 | <body>
15 |   <div></div><span>Double click the block</span>
16 | <script>
17 |   var divdbl = $("div:first");
18 |   divdbl.dblclick(function () {
19 |     divdbl.toggleClass('dbl');
20 |   });
21 | </script>
22 |
23 | </body>
24 | </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# deferred.always()

Categories: [Deferred Object](#)

```
deferred.always( alwaysCallbacks [, alwaysCallbacks ] )
```

Returns: [Deferred](#)

**Description:** *Add handlers to be called when the Deferred object is either resolved or rejected.*

```
deferred.always( alwaysCallbacks [, version added: 1.6  
alwaysCallbacks ] )
```

#### **alwaysCallbacks**

Type: [Function\(\)](#)

A function, or array of functions, that is called when the Deferred is resolved or rejected.

#### **alwaysCallbacks**

Type: [Function\(\)](#)

Optional additional functions, or arrays of functions, that are called when the Deferred is resolved or rejected.

The argument can be either a single function or an array of functions. When the Deferred is resolved or rejected, the `alwaysCallbacks` are called. Since `deferred.always()` returns the Deferred object, other methods of the Deferred object can be chained to this one, including additional `.always()` methods. When the Deferred is resolved or rejected, callbacks are executed in the order they were added, using the arguments provided to the `resolve`, `reject`, `resolveWith` or `rejectWith` method calls. For more information, see the documentation for [Deferred object](#).

## Example:

Since the `jQuery.get()` method returns a `jQueryXHR` object which is derived from a `Deferred` object we can attach a callback for both success and error using the `deferred.always()` method.

```
1 | $.get("test.php").always( function() {  
2 |     alert("$.get completed with success or error  
3 | } );
```

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# deferred.done()

Categories: [Deferred Object](#)

`deferred.done( doneCallbacks [, doneCallbacks ] )`

Returns: [Deferred](#)

**Description:** *Add handlers to be called when the Deferred object is resolved.*

`deferred.done( doneCallbacks [, doneCallbacks ] )`

version added: [1.5](#)

#### **doneCallbacks**

Type: [Function\(\)](#)

A function, or array of functions, that are called when the Deferred is resolved.

#### **doneCallbacks**

Type: [Function\(\)](#)

Optional additional functions, or arrays of functions, that are called when the Deferred is resolved.

The `deferred.done()` method accepts one or more arguments, all of which can be either a single function or an array of functions. When the Deferred is resolved, the doneCallbacks are called. Callbacks are executed in the order they were added. Since `deferred.done()` returns the deferred object, other methods of the deferred object can be chained to this one, including additional `.done()` methods. When the Deferred is resolved, doneCallbacks are executed using the arguments provided to the `resolve` or `resolveWith` method call in the order they were added. For more information, see the [documentation for Deferred object](#).

## Examples:

**Example:** Since the `jQuery.get` method returns a `jQueryXHR` object which is derived from a `Deferred` object we can attach a success callback using the `.done()` method.

```
1 | $.get("test.php").done(function() {
2 |     alert("$.get succeeded");
3 | });
```

**Example:** Resolve a `Deferred` object when the user clicks a button, triggering a number of callback functions:

```
1 | <!DOCTYPE html>
2 | <html>
3 | <head>
4 |     <script src="http://code.jquery.com/jquery-
5 | </head>
6 | <body>
7 |
8 |     <button>Go</button>
9 |     <p>Ready...</p>
10 |
11 | <script>
12 | /* 3 functions to call when the Deferred object
13 | function fn1() {
14 |     $("p").append(" 1 ");
15 | }
16 | function fn2() {
17 |     $("p").append(" 2 ");
18 | }
19 | function fn3(n) {
20 |     $("p").append(n + " 3 " + n);
```

```
21 }
22
23 /* create a deferred object */
24 var dfd = $.Deferred();
25
26 /* add handlers to be called when dfd is resolved */
27 dfd
28 /* .done() can take any number of functions or arrays of functions */
29 .done( [fn1, fn2], fn3, [fn2, fn1] )
30 /* we can chain done methods, too */
31 .done(function(n) {
32     $("p").append(n + " we're done.");
33 });
34
35 /* resolve the Deferred object when the button is clicked */
36 $("button").on("click", function() {
37     dfd.resolve("and");
38 });
39 </script>
40
41 </body>
42 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# deferred.fail()

Categories: [Deferred Object](#)

`deferred.fail( failCallbacks [, failCallbacks ] )`

Returns: [Deferred](#)

**Description:** *Add handlers to be called when the Deferred object is rejected.*

`deferred.fail( failCallbacks [, failCallbacks ] )`

version added: [1.5](#)

#### **failCallbacks**

Type: [Function\(\)](#)

A function, or array of functions, that are called when the Deferred is rejected.

#### **failCallbacks**

Type: [Function\(\)](#)

Optional additional functions, or arrays of functions, that are called when the Deferred is rejected.

The `deferred.fail()` method accepts one or more arguments, all of which can be either a single function or an array of functions. When the Deferred is rejected, the failCallbacks are called. Callbacks are executed in the order they were added. Since `deferred.fail()` returns the deferred object, other methods of the deferred object can be chained to this one, including additional `deferred.fail()` methods. The failCallbacks are executed using the arguments provided to the `deferred.reject()` or `deferred.rejectWith()` method call in the order they were added. For more information, see the documentation for [Deferred object](#).

## Example:

Since the `jQuery.get` method returns a `jQueryDeferred` object, which is derived from a `Deferred` you can attach a success and failure callback using the `deferred.done()` and `deferred.fail()` methods.

```
1 | $.get("test.php")
2 |   .done(function(){ alert("$.get succeeded");
3 |   .fail(function(){ alert("$.get failed!"); })
```

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# deferred.isRejected()

Categories: [Deferred Object](#)

`deferred.isRejected()`

Returns: [Boolean](#)

version deprecated: [1.7](#), removed: [1.8](#)

**Description:** *Determine whether a Deferred object has been rejected.*

`deferred.isRejected()`

version added: [1.5](#)

This method does not accept any arguments.

As of jQuery 1.7 this API has been deprecated; please use `deferred.state()` instead.

Returns `true` if the Deferred object is in the rejected state, meaning that either `deferred.reject()` or `deferred.rejectWith()` has been called for the object and the failCallbacks have been called (or are in the process of being called).

Note that a Deferred object can be in one of three states: pending, resolved, or rejected; use `deferred.isResolved()` to determine whether the Deferred object is in the resolved state. These methods are primarily useful for debugging, for example to determine whether a Deferred has already been resolved even though you are inside code that intended to reject it.

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# deferred.isResolved()

Categories: [Deferred Object](#)

`deferred.isResolved()`

Returns: [Boolean](#)

version deprecated: [1.7](#), removed: [1.8](#)

**Description:** *Determine whether a Deferred object has been resolved.*

`deferred.isResolved()`

version added: [1.5](#)

This method does not accept any arguments.

This API is `deprecated` as of jQuery 1.7 and `removed` as of jQuery 1.8; please use `deferred.state()` instead.

Returns `true` if the Deferred object is in the resolved state, meaning that either `deferred.resolve()` or `deferred.resolveWith()` has been called for the object and the doneCallbacks have been called (or are in the process of being called).

Note that a Deferred object can be in one of three states: pending, resolved, or rejected; use `deferred.isRejected()` to determine whether the Deferred object is in the rejected state. These methods are primarily useful for debugging, for example to determine whether a Deferred has already been resolved even though you are inside code that intended to reject it.

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# deferred.notify()

Categories: [Deferred Object](#)

`deferred.notify( args )`

Returns: [Deferred](#)

**Description:** *Call the `progressCallbacks` on a `Deferred` object with the given `args`.*

`deferred.notify( args )`

version added: [1.7](#)

**args**

Type: [Object](#)

Optional arguments that are passed to the `progressCallbacks`.

Normally, only the creator of a `Deferred` should call this method; you can prevent other code from changing the `Deferred`'s state or reporting status by returning a restricted `Promise` object through `deferred.promise()`.

When `deferred.notify` is called, any `progressCallbacks` added by `deferred.then` or `deferred.progress` are called. Callbacks are executed in the order they were added. Each callback is passed the `args` from the `.notify()`. Any calls to `.notify()` after a `Deferred` is resolved or rejected (or any `progressCallbacks` added after that) are ignored. For more information, see the documentation for [Deferred object](#).

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# deferred.notifyWith()

Categories: [Deferred Object](#)

`deferred.notifyWith( context [, args ] )`

Returns: [Deferred](#)

**Description:** *Call the progressCallbacks on a Deferred object with the given context and `args`.*

`deferred.notifyWith( context [, args ] )` version added: [1.7](#)

**context**

Type: [Object](#)

Context passed to the progressCallbacks as the `this` object.

**args**

Type: [Object](#)

Optional arguments that are passed to the progressCallbacks.

Normally, only the creator of a Deferred should call this method; you can prevent other code from changing the Deferred's state or reporting status by returning a restricted Promise object through `deferred.promise()`.

When `deferred.notifyWith` is called, any progressCallbacks added by `deferred.then` or `deferred.progress` are called. Callbacks are executed in the order they were added. Each callback is passed the `args` from the `.notifyWith()`. Any calls to `.notifyWith()` after a Deferred is resolved or rejected (or any progressCallbacks added after that) are ignored. For more information, see the documentation for [Deferred object](#).

POWERED BY HERONOTE

A new version of this book is available!

---

# deferred.pipe()

Categories: [Deferred Object](#)

`deferred.pipe( [doneFilter ] [, failFilter ] )` Returns: [Promise](#)  
version deprecated: [1.8](#)

**Description:** *Utility method to filter and/or chain Deferreds.*

`deferred.pipe( [doneFilter ] [, failFilter ] )` version added: [1.6](#)

**doneFilter**

Type: [Function\(\)](#)

An optional function that is called when the Deferred is resolved.

**failFilter**

Type: [Function\(\)](#)

An optional function that is called when the Deferred is rejected.

`deferred.pipe( [doneFilter ] [, failFilter ] [, progressFilter ] )` version added: [1.7](#)

**doneFilter**

Type: [Function\(\)](#)

An optional function that is called when the Deferred is resolved.

**failFilter**

Type: [Function\(\)](#)

An optional function that is called when the Deferred is rejected.

**progressFilter**

Type: [Function\(\)](#)

An optional function that is called when progress notifications are sent to the Deferred.

**Deprecation Notice:** As of jQuery 1.8, the `deferred.pipe()` method is deprecated. The `deferred.then()` method, which replaces it, should be used instead.

The `deferred.pipe()` method returns a new promise that filters the

status and values of a deferred through a function. The `doneFilter` and `failFilter` functions filter the original deferred's resolved / rejected status and values. **As of jQuery 1.7**, the method also accepts a `progressFilter` function to filter any calls to the original deferred's `notify` or `notifyWith` methods. These filter functions can return a new value to be passed along to the piped promise's `done()` or `fail()` callbacks, or they can return another observable object (Deferred, Promise, etc) which will pass its resolved / rejected status and values to the piped promise's callbacks. If the filter function used is `null`, or not specified, the piped promise will be resolved or rejected with the same values as the original.

## Examples:

### Example: *Filter resolve value:*

```
1 | var defer = $.Deferred(),
2 |   filtered = defer.pipe(function( value ) {
3 |     return value * 2;
4 |   });
5 |
6 | defer.resolve( 5 );
7 | filtered.done(function( value ) {
8 |   alert( "Value is ( 2*5 = ) 10: " + value );
9 | });
```

### Example: *Filter reject value:*

```
1 | var defer = $.Deferred(),
2 |   filtered = defer.pipe( null, function( value ) {
3 |     return value * 3;
4 |   });
5 |
6 | defer.reject( 6 );
7 | filtered.fail(function( value ) {
8 |   alert( "Value is ( 3*6 = ) 18: " + value );
9 | });
```

### Example: *Chain tasks:*

```
1 | var request = $.ajax( url1, { dataType: "json" } );
2 |   chained = request.pipe(function( data ) {
3 |     return $.ajax( url2, { data: { user: data } } );
```

```
4     });  
5  
6     chained.done(function( data ) {  
7         // data retrieved from url2 as provided by the first r  
8     });  
9
```

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# deferred.progress()

Categories: [Deferred Object](#)

`deferred.progress( progressCallbacks )` Returns: [Deferred](#)

**Description:** *Add handlers to be called when the Deferred object generates progress notifications.*

**deferred.progress(  
progressCallbacks )**

version added: [1.7](#)

**progressCallbacks**

Type: [Function\(\)](#)

A function, or array of functions, that is called when the Deferred generates progress notifications.

The argument can be either a single function or an array of functions. When the Deferred generates progress notifications by calling `notify` or `notifyWith`, the `progressCallbacks` are called. Since `deferred.progress()` returns the Deferred object, other methods of the Deferred object can be chained to this one. When the Deferred is resolved or rejected, progress callbacks will no longer be called, with the exception that any `progressCallbacks` added after the Deferred enters the resolved or rejected state are executed immediately when they are added, using the arguments that were passed to the `.notify()` or `notifyWith()` call. For more information, see the documentation [jQuery.Deferred\(\)](#).

POWERED BY HERONOTE

A new version of this book is available!

---

# deferred.promise()

Categories: [Deferred Object](#)

`deferred.promise( [target ] )`

Returns: [Promise](#)

**Description:** *Return a Deferred's Promise object.*

`deferred.promise( [target ] )`

version added: [1.5](#)

**target**

Type: [Object](#)

Object onto which the promise methods have to be attached

The `deferred.promise()` method allows an asynchronous function to prevent other code from interfering with the progress or status of its internal request. The Promise exposes only the Deferred methods needed to attach additional handlers or determine the state (`then`, `done`, `fail`, `always`, `pipe`, `progress`, and `state`), but not ones that change the state (`resolve`, `reject`, `notify`, `resolveWith`, `rejectWith`, and `notifyWith`).

If `target` is provided, `deferred.promise()` will attach the methods onto it and then return this object rather than create a new one. This can be useful to attach the Promise behavior to an object that already exists.

If you are creating a Deferred, keep a reference to the Deferred so that it can be resolved or rejected at some point. Return *only* the Promise object via `deferred.promise()` so other code can register callbacks or inspect the current state.

For more information, see the documentation for [Deferred object](#).

## Examples:

**Example:** Create a Deferred and set two timer-based functions to either resolve or reject the Deferred after a random interval. Whichever one fires first "wins" and will call one of the callbacks. The second timeout has no effect since the Deferred is already complete (in a resolved or rejected state) from the first timeout action. Also set a timer-based progress notification function, and call a progress handler that adds "working..." to the document body.

```
1 function asyncEvent(){
2     var dfd = new jQuery.Deferred();
3
4     // Resolve after a random interval
5     setTimeout(function(){
6         dfd.resolve("hurray");
7     }, Math.floor(400+Math.random()*2000));
8
9     // Reject after a random interval
10    setTimeout(function(){
11        dfd.reject("sorry");
12    }, Math.floor(400+Math.random()*2000));
13
14    // Show a "working..." message every half
15    setTimeout(function working(){
16        if ( dfd.state() === "pending" ) {
17            dfd.notify("working... ");
18            setTimeout(working, 500);
19        }
20    }, 1);
21
22    // Return the Promise so caller can't cha
23    return dfd.promise();
```

```

24 }
25
26 // Attach a done, fail, and progress handler for the asy
27 $.when( asyncEvent() ).then(
28     function(status){
29         alert( status+' , things are going well'
30     },
31     function(status){
32         alert( status+' , you fail this time' );
33     },
34     function(status){
35         $("body").append(status);
36     }
37 );

```

**Example** · Use the *target* argument to promote an existing object to a Promise:

```

1 // Existing object
2 var obj = {
3     hello: function( name ) {
4         alert( "Hello " + name );
5     }
6 },
7 // Create a Deferred
8 defer = $.Deferred();
9
10 // Set object as a promise
11 defer.promise( obj );
12
13 // Resolve the deferred
14 defer.resolve( "John" );
15
16 // Use the object as a Promise
17 obj.done(function( name ) {

```

```
18 | obj.hello( name ); // will alert "Hello John"  
19 | }).hello( "Karl" ); // will alert "Hello Karl"
```

POWERED BY **HERONOTE**

A new version of this book is available!

---

# deferred.reject()

Categories: [Deferred Object](#)

`deferred.reject( args )`

Returns: [Deferred](#)

**Description:** *Reject a Deferred object and call any failCallbacks with the given `args`.*

`deferred.reject( args )`

version added: [1.5](#)

**args**

Type: [Object](#)

Optional arguments that are passed to the failCallbacks.

Normally, only the creator of a Deferred should call this method; you can prevent other code from changing the Deferred's state by returning a restricted Promise object through `deferred.promise()`.

When the Deferred is rejected, any failCallbacks added by `deferred.then` or `deferred.fail` are called. Callbacks are executed in the order they were added. Each callback is passed the `args` from the `deferred.reject()` call. Any failCallbacks added after the Deferred enters the rejected state are executed immediately when they are added, using the arguments that were passed to the `.reject()` call. For more information, see the documentation for [Deferred object](#).

**POWERED BY HERONOTE**

A new version of this book is available!

---

# deferred.rejectWith()

Categories: [Deferred Object](#)

`deferred.rejectWith( context [, args ] )`

Returns: [Deferred](#)

**Description:** *Reject a Deferred object and call any failCallbacks with the given `context` and `args`.*

`deferred.rejectWith( context [, args ] )` version added: [1.5](#)

**context**

Type: [Object](#)

Context passed to the failCallbacks as the `this` object.

**args**

Type: [Array](#)

An optional array of arguments that are passed to the failCallbacks.

Normally, only the creator of a Deferred should call this method; you can prevent other code from changing the Deferred's state by returning a restricted Promise object through `deferred.promise()`.

When the Deferred is rejected, any failCallbacks added by `deferred.then` or `deferred.fail` are called. Callbacks are executed in the order they were added. Each callback is passed the `args` from the `deferred.reject()` call. Any failCallbacks added after the Deferred enters the rejected state are executed immediately when they are added, using the arguments that were passed to the `.reject()` call. For more information, see the documentation for [Deferred object](#).

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# deferred.resolve()

Categories: [Deferred Object](#)

`deferred.resolve( args )`

Returns: [Deferred](#)

**Description:** *Resolve a Deferred object and call any doneCallbacks with the given `args`.*

**deferred.resolve( args )**

version added: [1.5](#)

**args**

Type: [Object](#)

Optional arguments that are passed to the doneCallbacks.

When the Deferred is resolved, any doneCallbacks added by `deferred.then` or `deferred.done` are called. Callbacks are executed in the order they were added. Each callback is passed the `args` from the `.resolve()`. Any doneCallbacks added after the Deferred enters the resolved state are executed immediately when they are added, using the arguments that were passed to the `.resolve()` call. For more information, see the documentation for [Deferred object](#).

POWERED BYHERONOTE

A new version of this book is available!

---

# deferred.resolveWith()

Categories: [Deferred Object](#)

`deferred.resolveWith( context [, args ] )` Returns: [Deferred](#)

**Description:** *Resolve a Deferred object and call any doneCallbacks with the given `context` and `args`.*

`deferred.resolveWith( context [, args ] )` version added: [1.5](#)

**context**

Type: [Object](#)

Context passed to the doneCallbacks as the `this` object.

**args**

Type: [Array](#)

An optional array of arguments that are passed to the doneCallbacks.

Normally, only the creator of a Deferred should call this method; you can prevent other code from changing the Deferred's state by returning a restricted Promise object through `deferred.promise()`.

When the Deferred is resolved, any doneCallbacks added by `deferred.then` or `deferred.done` are called. Callbacks are executed in the order they were added. Each callback is passed the `args` from the `.resolve()`. Any doneCallbacks added after the Deferred enters the resolved state are executed immediately when they are added, using the arguments that were passed to the `.resolve()` call. For more information, see the documentation for [Deferred object](#).

POWERED BYHERONOTE

A new version of this book is available!

---

# deferred.state()

Categories: [Deferred Object](#)

`deferred.state()`

Returns: `String`

**Description:** *Determine the current state of a Deferred object.*

`deferred.state()`

version added: `1.7`

This method does not accept any arguments.

The `deferred.state()` method returns a string representing the current state of the Deferred object. The Deferred object can be in one of three states:

**"pending"**: The Deferred object is not yet in a completed state (neither "rejected" nor "resolved").

**"resolved"**: The Deferred object is in the resolved state, meaning that either `deferred.resolve()` or `deferred.resolveWith()` has been called for the object and the doneCallbacks have been called (or are in the process of being called).

**"rejected"**: The Deferred object is in the rejected state, meaning that either `deferred.reject()` or `deferred.rejectWith()` has been called for the object and the failCallbacks have been called (or are in the process of being called).

This method is primarily useful for debugging to determine, for example, whether a Deferred has already been resolved even though you are inside code that intended to reject it.

**A new version of this book is available!**

---

# deferred.then()

Categories: [Deferred Object](#)

```
deferred.then( doneFilter [, failFilter ] [, progressFilter ] )  
Returns: Promise
```

**Description:** *Add handlers to be called when the Deferred object is resolved, rejected, or still in progress.*

```
deferred.then( doneFilter [, failFilter ] version added: 1.8  
[, progressFilter ] )
```

#### **doneFilter**

Type: [Function\(\)](#)

A function that is called when the Deferred is resolved.

#### **failFilter**

Type: [Function\(\)](#)

An optional function that is called when the Deferred is rejected.

#### **progressFilter**

Type: [Function\(\)](#)

An optional function that is called when progress notifications are sent to the Deferred.

```
deferred.then( version added: 1.5, removed: 1.8  
doneCallbacks, failCallbacks )
```

#### **doneCallbacks**

Type: [Function\(\)](#)

A function, or array of functions, called when the Deferred is resolved.

#### **failCallbacks**

Type: [Function\(\)](#)

A function, or array of functions, called when the Deferred is rejected.

```
deferred.then( version added: 1.7, removed: 1.8  
doneCallbacks, failCallbacks [, progressCallbacks ] )
```

#### **doneCallbacks**

Type: `Function()`

A function, or array of functions, called when the Deferred is resolved.

### **failCallbacks**

Type: `Function()`

A function, or array of functions, called when the Deferred is rejected.

### **progressCallbacks**

Type: `Function()`

A function, or array of functions, called when the Deferred notifies progress.

**Prior to jQuery 1.8**, the arguments could be a function or an array of functions.

For all signatures, the arguments can be `null` if no callback of that type is desired. Alternatively, use `.done()`, `.fail()` or `.progress()` to set only one type of callback without filtering status or values.

**As of jQuery 1.8**, the `deferred.then()` method returns a new promise that can filter the status and values of a deferred through a function, replacing the now-deprecated `deferred.pipe()` method. The `doneFilter` and `failFilter` functions filter the original deferred's resolved / rejected status and values. The `progressFilter` function filters any calls to the original deferred's `notify` or `notifyWith` methods. These filter functions can return a new value to be passed along to the promise's `.done()` or `.fail()` callbacks, or they can return another observable object (Deferred, Promise, etc) which will pass its resolved / rejected status and values to the promise's callbacks. If the filter function used is `null`, or not specified, the promise will be resolved or rejected with the same values as the original.

Callbacks are executed in the order they were added. Since `deferred.then` returns a Promise, other methods of the Promise object can be chained to this one, including additional `.then()` methods.

## Examples:

**Example:** Since the `jQuery.get` method returns a `jQueryXHR` object which is derived from a `Deferred` object, we can attach handlers using the `.then` method.

```
1 | $.get("test.php").then(  
2 |     function(){ alert("$.get succeeded"); },  
3 |     function(){ alert("$.get failed!"); }  
4 | );
```

**Example:** Filter the resolve value:

```
1 | <!DOCTYPE html>  
2 | <html>  
3 | <head>  
4 |     <script src="http://code.jquery.com/jquery->  
5 | </head>  
6 | <body>  
7 |  
8 | <button>Filter Resolve</button>  
9 | <p></p>  
10 |  
11 | <script>  
12 | var filterResolve = function() {  
13 |  
14 |     var defer = $.Deferred(),  
15 |         filtered = defer.then(function( value )  
16 |             return value * 2;  
17 |         });  
18 |  
19 |     defer.resolve( 5 );  
20 |     filtered.done(function( value ) {  
21 |         $( "p" ).html( "Value is ( 2*5 = ) 10: " )
```

```

22     });
23 };
24
25 $( "button" ).on( "click", filterResolve );
26 </script>
27
28 </body>
29 </html>

```

## Demo

### Example: *Filter reject value:*

```

1  var defer = $.Deferred(),
2     filtered = defer.then( null, function( value ) {
3     return value * 3;
4     });
5
6  defer.reject( 6 );
7  filtered.fail(function( value ) {
8     alert( "Value is ( 3*6 = ) 18: " + value );
9  });

```

### Example: *Chain tasks:*

```

1  var request = $.ajax( url1, { dataType: "json" } );
2     chained = request.then(function( data ) {
3     return $.ajax( url2, { data: { user: data } } );
4     });
5
6  chained.done(function( data ) {
7     // data retrieved from url2 as provided by the
8  });
9

```

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# .delay()

Categories: [Effects](#) > [Custom](#)

`.delay( duration [, queueName ] )`

Returns: [jQuery](#)

**Description:** *Set a timer to delay execution of subsequent items in the queue.*

`.delay( duration [, queueName ] )`

version added: [1.4](#)

**duration**

Type: [Integer](#)

An integer indicating the number of milliseconds to delay execution of the next item in the queue.

**queueName**

Type: [String](#)

A string containing the name of the queue. Defaults to `fx`, the standard effects queue.

Added to jQuery in version 1.4, the `.delay()` method allows us to delay the execution of functions that follow it in the queue. It can be used with the standard effects queue or with a custom queue. Only subsequent events in a queue are delayed; for example this will *not* delay the no-arguments forms of `.show()` or `.hide()` which do not use the effects queue.

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings `'fast'` and `'slow'` can be supplied to indicate durations of 200 and 600 milliseconds, respectively.

Using the standard effects queue, we can, for example, set an 800-millisecond delay between the `.slideUp()` and `.fadeIn()` of `<div id="foo">`:

```
1 | $('#foo').slideUp(300).delay(800).fadeIn(400);
```

When this statement is executed, the element slides up for 300 milliseconds and then pauses for 800 milliseconds before fading in

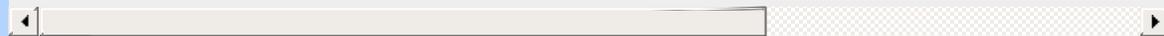
for 400 milliseconds.

**The `.delay()` method is best for delaying between queued jQuery effects. Because it is limited—it doesn't, for example, offer a way to cancel the delay—`.delay()` is not a replacement for JavaScript's native `setTimeout` function, which may be more appropriate for certain use cases.**

## Example:

*Animate the hiding and showing of two divs, delaying the first before showing it.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5  div { position: absolute; width: 60px; height: 60px;
6  .first { background-color: #3f3; left: 0;}
7  .second { background-color: #33f; left: 80px;}
8  </style>
9    <script src="http://code.jquery.com/jquery-1.10.2.js"></script>
10 </head>
11 <body>
12
13 <p><button>Run</button></p>
14 <div class="first"></div>
15 <div class="second"></div>
16 <script>
17   $("button").click(function() {
18     $(".first").slideUp(300).delay(800).slideDown(300);
19     $(".second").slideUp(300).fadeIn(400);
20   });</script>
21
22 </body>
23 </html>
```



Demo

**A new version of this book is available!**

---

# .delegate()

Categories: [Events](#) > [Event Handler Attachment](#)

`.delegate( selector, eventType, handler(eventObject) )`

Returns: [jQuery](#)

**Description:** *Attach a handler to one or more events for all elements that match the selector, now or in the future, based on a specific set of root elements.*

**.delegate( selector, eventType, handler(eventObject) )**

version added: [1.4.2](#)

**selector**

Type: [String](#)

A selector to filter the elements that trigger the event.

**eventType**

Type: [String](#)

A string containing one or more space-separated JavaScript event types, such as "click" or "keydown," or custom event names.

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute at the time the event is triggered.

**.delegate( selector, eventType, eventData, handler(eventObject) )**

version added: [1.4.2](#)

**selector**

Type: [String](#)

A selector to filter the elements that trigger the event.

**eventType**

Type: [String](#)

A string containing one or more space-separated JavaScript event types, such as "click" or "keydown," or custom event names.

**eventData**

Type: [Object](#)

An object containing data that will be passed to the event

handler.

### handler(eventObject)

Type: [Function\(\)](#)

A function to execute at the time the event is triggered.

## **.delegate( selector, events )**

version added: [1.4.3](#)

### selector

Type: [String](#)

A selector to filter the elements that trigger the event.

### events

Type: [PlainObject](#)

A plain object of one or more event types and functions to execute for them.

As of jQuery 1.7, `.delegate()` has been superseded by the `.on()` method. For earlier versions, however, it remains the most effective means to use event delegation. More information on event binding and delegation is in the `.on()` method. In general, these are the equivalent templates for the two methods:

```
1 | // jQuery 1.4.3+
2 | $(elements).delegate( selector, events, data,
3 | // jQuery 1.7+
4 | $(elements).on( events, selector, data, handle
```

For example, the following `.delegate()` code:

```
1 | $("table").delegate("td", "click", function()
2 |     $(this).toggleClass("chosen");
3 | });
```

is equivalent to the following code written using `.on()`:

```
1 | $("table").on("click", "td", function() {  
2 |     $(this).toggleClass("chosen");  
3 | });
```

To remove events attached with `delegate()`, see the [.undelegate\(\)](#) method.

Passing and handling event data works the same way as it does for `.on()`.

## Additional Notes:

Since the `.live()` method handles events once they have propagated to the top of the document, it is not possible to stop propagation of live events. Similarly, events handled by `.delegate()` will propagate to the elements to which they are delegated; event handlers bound on any elements below it in the DOM tree will already have been executed by the time the delegated event handler is called. These handlers, therefore, may prevent the delegated handler from triggering by calling `event.stopPropagation()` or returning `false`.

## Examples:

**Example:** Click a paragraph to add another. Note that `delegate()` attaches a click event handler to all paragraphs - even new ones.

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5      p { background:yellow; font-weight:bold; cu
6         padding:5px; }
7      p.over { background: #ccc; }
8      span { color:red; }
9    </style>
10   <script src="http://code.jquery.com/jquery-
11 </head>
12 <body>
13   <p>Click me!</p>
14
15   <span></span>
16 <script>
17   $("body").delegate("p", "click", function
18     $(this).after("<p>Another paragraph!</p>
19     });
20 </script>
21
22 </body>
23 </html>
```

Demo

**Example:** To display each paragraph's text in an alert box whenever it is clicked:

```
1 | $("body").delegate("p", "click", function(){
2 |     alert( $(this).text() );
3 | });
```

**Example:** To cancel a default action and prevent it from bubbling up, return false:

```
1 | $("body").delegate("a", "click", function() {
```

**Example:** To cancel only the default action by using the `preventDefault` method.

```
1 | $("body").delegate("a", "click", function(event)
2 |     event.preventDefault();
3 | });
```

**Example:** Can bind custom events too.

```
1 | <!DOCTYPE html>
2 | <html>
3 | <head>
4 |     <style>
5 |     p { color:red; }
6 |     span { color:blue; }
7 |     </style>
8 |     <script src="http://code.jquery.com/jquery-1.10.2.js"></script>
9 | </head>
10 | <body>
11 |     <p>Has an attached custom event.</p>
12 |     <button>Trigger custom event</button>
13 |     <span style="display:none;"></span>
```

```
14 <script>
15
16     $("body").delegate("p", "myCustomEvent", function
17     $(this).text("Hi there!");
18     $("span").stop().css("opacity", 1)
19         .text("myName = " + myName)
20         .fadeIn(30).fadeOut(1000);
21     });
22     $("button").click(function () {
23         $("p").trigger("myCustomEvent");
24     });
25
26 </script>
27
28 </body>
29 </html>
```

Demo

POWERED BY HERONOTE

A new version of this book is available!

---

# .dequeue()

Categories: [Effects](#) > [Custom](#) | [Data](#) | [Utilities](#)

`.dequeue( [queueName ] )`

Returns: [jQuery](#)

**Description:** *Execute the next function on the queue for the matched elements.*

`.dequeue( [queueName ] )`

version added: [1.2](#)

**queueName**

Type: [String](#)

A string containing the name of the queue. Defaults to `fx`, the standard effects queue.

When `.dequeue()` is called, the next function on the queue is removed from the queue, and then executed. This function should in turn (directly or indirectly) cause `.dequeue()` to be called, so that the sequence can continue.

## Example:

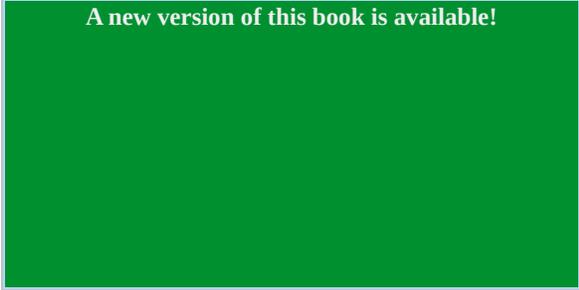
*Use `dequeue` to end a custom queue function which allows the queue to keep going.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     div { margin:3px; width:50px; position:abso.
6     height:50px; left:10px; top:30px;
7     background-color:yellow; }
8     div.red { background-color:red; }
9   </style>
10  <script src="http://code.jquery.com/jquery-
11 </head>
12 <body>
13   <button>Start</button>
14 <div></div>
15 <script>
16 $("button").click(function () {
17   $("div").animate({left:'+=200px'}, 2000);
18   $("div").animate({top:'0px'}, 600);
19   $("div").queue(function () {
20     $(this).toggleClass("red");
21     $(this).dequeue();
22   });
23   $("div").animate({left:'10px', top:'30px'},
24 });
25 </script>
26
27 </body>
28 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

# Descendant Selector (“ancestor descendant”)

Categories: [Selectors](#) > [Hierarchy](#)

## descendant selector

**Description:** *Selects all elements that are descendants of a given ancestor.*

**jQuery( "ancestor descendant" )**

version added: [1.0](#)

**ancestor:** Any valid selector.

**descendant:** A selector to filter the descendant elements.

A descendant of an element could be a child, grandchild, great-grandchild, and so on, of that element.

## Example:

*Finds all input descendants of forms.*

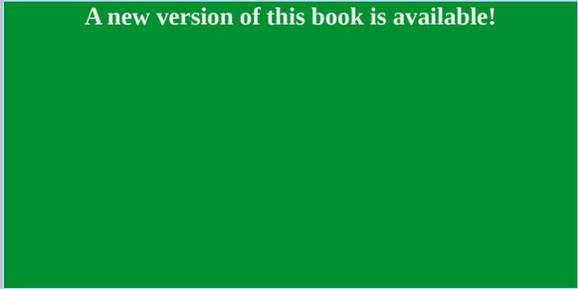
```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5
6   body { font-size:14px; }
7   form { border:2px green solid; padding:2px;
8         background:#efe; }
9   div { color:red; }
10  fieldset { margin:1px; padding:3px; }
11 </style>
12 <script src="http://code.jquery.com/jquery-
13 </head>
14 <body>
15   <form>
16     <div>Form is surrounded by the green outl
17     <label>Child:</label>
18     <input name="name" />
19
20     <fieldset>
21       <label>Grandchild:</label>
22       <input name="newsletter" />
23     </fieldset>
24
25   </form>
26   Sibling to form: <input name="none" />
27 <script>$("form input").css("border", "2px do
28
29 </body>
30 </html>
```



Demo

POWERED BY **HERONOTE**

A new version of this book is available!



---

# .detach()

Categories: [Manipulation](#) > [DOM Removal](#)

`.detach( [selector ] )`

Returns: [jQuery](#)

**Description:** *Remove the set of matched elements from the DOM.*

`.detach( [selector ] )`

version added: [1.4](#)

**selector**

Type: [Selector](#)

A selector expression that filters the set of matched elements to be removed.

The `.detach()` method is the same as `.remove()`, except that `.detach()` keeps all jQuery data associated with the removed elements. This method is useful when removed elements are to be reinserted into the DOM at a later time.

## Example:

*Detach all paragraphs from the DOM*

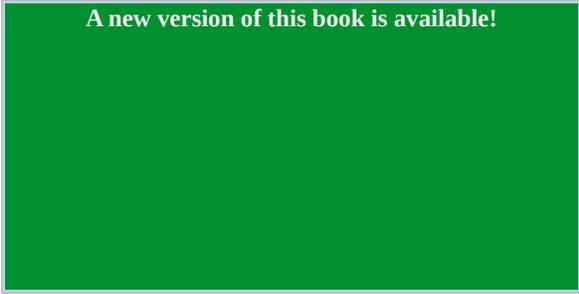
```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>p { background:yellow; margin:6px 0;
5   <script src="http://code.jquery.com/jquery-
6 </head>
7 <body>
8   <p>Hello</p>
9   how are
10  <p>you?</p>
11  <button>Attach/detach paragraphs</button>
12 <script>
13   $("p").click(function(){
14     $(this).toggleClass("off");
15   });
16   var p;
17   $("button").click(function(){
18     if ( p ) {
19       p.appendTo("body");
20       p = null;
21     } else {
22       p = $("p").detach();
23     }
24   });</script>
25
26 </body>
27 </html>
```

Demo

---

POWERED BY **HERONOTE**

A new version of this book is available!



---

# .die()

Categories: [Events](#) > [Event Handler Attachment](#)

`.die()`

Returns: [jQuery](#)

version deprecated: [1.7](#), removed: [1.9](#)

**Description:** Remove event handlers previously attached using `.live()` from the elements.

`.die()`

version added: [1.4.1](#)

This method does not accept any arguments.

`.die( eventType [, handler ] )`

version added: [1.3](#)

**eventType**

Type: [String](#)

A string containing a JavaScript event type, such as `click` or `keydown`.

**handler**

Type: [String](#)

The function that is no longer to be executed.

`.die( events )`

version added: [1.4.3](#)

**events**

Type: [PlainObject](#)

A plain object of one or more event types, such as `click` or `keydown` and their corresponding functions that are no longer to be executed.

Any handler that has been attached with `.live()` can be removed with `.die()`. This method is analogous to calling `.off()` with no arguments, which is used to remove all handlers attached with `.on()`. See the discussions of `.live()` and `.off()` for further details.

If used without an argument, `.die()` removes *all* event handlers previously attached using `.live()` from the elements.

**As of jQuery 1.7**, use of `.die()` (and its complementary method, `.live()`) is not recommended. Instead, use `.off()` to remove event handlers bound with `.on()`

**Note:** In order for `.die()` to function correctly, the selector used with it must match exactly the selector initially used with `.live()`.

## Examples:

**Example:** To unbind all live events from all paragraphs, write:

```
1 | $("p").die()
```

**Example:** To unbind all live click events from all paragraphs, write:

```
1 | $("p").die("click")
```

**Example:** To unbind just one previously bound handler, pass the function in as the second argument:

```
1 | var foo = function () {  
2 |     // code to handle some kind of event  
3 | };  
4 |  
5 | $("p").live("click", foo); // ... now foo will  
6 |  
7 | $("p").die("click", foo); // ... foo will no l
```

**A new version of this book is available!**

---

# :disabled Selector

Categories: [Selectors](#) > [Form](#)

## disabled selector

**Description:** *Selects all elements that are disabled.*

**jQuery( ":disabled" )**

version added: [1.0](#)

As with other pseudo-class selectors (those that begin with a ":"), it is recommended to precede it with a tag name or some other selector; otherwise, the universal selector ("\*") is implied. In other words, the bare `$(':disabled')` is equivalent to `$('*:disabled')`, so `$('input:disabled')` or similar should be used instead.

Although their resulting selections are usually the same, the `:disabled` selector is subtly different from the `[disabled]` attribute selector; `:disabled` checks the boolean (true/false) value of the element's disabled property while `[disabled]` checks for the existence of the disabled attribute.

The `:disabled` selector should only be used for selecting HTML elements that support the `disabled` attribute (`<button>`, `<input>`, `<optgroup>`, `<option>`, `<select>`, and `<textarea>`).

## Example:

*Finds all input elements that are disabled.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
5  </head>
6  <body>
7    <form>
8
9      <input name="email" disabled="disabled" type="text" value="Email" />
10     <input name="id" type="text" value="ID" />
11   </form>
12   <script>$("#input:disabled").val("this is it");
13
14 </body>
15 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# .each()

Categories: [Miscellaneous](#) > [Collection Manipulation](#) | [Traversing](#)

`.each( function(index, Element) )`

Returns: [jQuery](#)

**Description:** *Iterate over a jQuery object, executing a function for each matched element.*

`.each( function(index, Element) )`

version added: [1.0](#)

**function(index, Element)**

Type: [Function\(\)](#)

A function to execute for each matched element.

The `.each()` method is designed to make DOM looping constructs concise and less error-prone. When called it iterates over the DOM elements that are part of the jQuery object. Each time the callback runs, it is passed the current loop iteration, beginning from 0. More importantly, the callback is fired in the context of the current DOM element, so the keyword `this` refers to the element.

Suppose you have a simple unordered list on the page:

```
1 | <ul>
2 |   <li>foo</li>
3 |   <li>bar</li>
4 | </ul>
```

You can select the list items and iterate across them:

```
1 | $("li").each(function( index ) {
2 |   console.log( index + ": " + $(this).text()
3 | });
```

A message is thus logged for each item in the list:

0: foo 1: bar

You can stop the loop from within the callback function by returning `false`.

Note: most jQuery methods that return a jQuery object also loop through the set of elements in the jQuery collection — a process known as *implicit iteration*. When this occurs, it is often unnecessary to *explicitly* iterate with the `.each()` method:

```
1 // The .each() method is unnecessary here:
2 $( "li" ).each(function() {
3     $(this).addClass( "foo" );
4 });
5
6 // Instead, you should rely on implicit iteration:
7 $( "li" ).addClass( "bar" );
```

## Examples:

**Example:** Iterate over three divs and sets their color property.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     div { color:red; text-align:center; cursor:|
6           font-weight:bolder; width:300px; }
7   </style>
8   <script src="http://code.jquery.com/jquery-
9 </head>
10 <body>
11   <div>Click here</div>
12   <div>to iterate through</div>
13   <div>these divs.</div>
14 <script>
15   $(document.body).click(function () {
16     $("div").each(function (i) {
17       if ( this.style.color != "blue" ) {
18         this.style.color = "blue";
19       } else {
20         this.style.color = "";
21       }
22     });
23   });
24 </script>
25
26 </body>
27 </html>
```

## Demo

**Example:** To access a jQuery object instead of the regular DOM element, use `$(this)`. For example:

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5      ul { font-size:18px; margin:0; }
6      span { color:blue; text-decoration:underline; }
7      .example { font-style:italic; }
8    </style>
9    <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
10 </head>
11 <body>
12   To do list: <span>(click here to change)</span>
13   <ul>
14     <li>Eat</li>
15     <li>Sleep</li>
16
17     <li>Be merry</li>
18   </ul>
19 <script>
20   $( "span" ).click(function () {
21     $( "li" ).each(function(){
22       $( this ).toggleClass( "example" );
23     });
24   });
25
26 </script>
27
28 </body>
29 </html>
```

## Demo

**Example:** Use "return" to break out of each() loops early.

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5      div { width:40px; height:40px; margin:5px;
6            border:2px blue solid; text-align:center; }
7      span { color:red; }
8    </style>
9    <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
10 </head>
11 <body>
12   <button>Change colors</button>
13   <span></span>
14   <div></div>
15   <div></div>
16
17   <div></div>
18   <div></div>
19   <div id="stop">Stop here</div>
20   <div></div>
21
22   <div></div>
23   <div></div>
24 <script>
25   $( "button" ).click(function () {
26     $( "div" ).each(function ( index, domEle ) {
27       // domEle == this
28       $( domEle ).css( "backgroundColor", "red" );
29       if ( $(this).is( "#stop" ) ) {
30         $( "span" ).text( "Stopped at div id: " + index );
31         return false;
32       }
33     });
```

```
34     });  
35  
36 </script>  
37  
38 </body>  
39 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# Element Selector (“element”)

Categories: [Selectors](#) > [Basic](#)

## element selector

**Description:** *Selects all elements with the given tag name.*

**jQuery( "element" )**

version added: [1.0](#)

**element:** An element to search for. Refers to the tagName of DOM nodes.

JavaScript's `getElementsByName()` function is called to return the appropriate elements when this expression is used.

## Example:

*Finds every DIV element.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5      div, span {
6        width: 60px;
7        height: 60px;
8        float:left;
9        padding: 10px;
10       margin: 10px;
11       background-color: #EEEEEE;
12     }
13   </style>
14   <script src="http://code.jquery.com/jquery-
15 </head>
16 <body>
17   <div>DIV1</div>
18
19   <div>DIV2</div>
20   <span>SPAN</span>
21   <script>$("div").css("border", "9px solid red"
22
23 </body>
24 </html>
```

Demo

**A new version of this book is available!**

---

# .empty()

Categories: [Manipulation](#) > [DOM Removal](#)

`.empty()`

Returns: [jQuery](#)

**Description:** *Remove all child nodes of the set of matched elements from the DOM.*

`.empty()`

version added: [1.0](#)

This method does not accept any arguments.

This method removes not only child (and other descendant) elements, but also any text within the set of matched elements. This is because, according to the DOM specification, any string of text within an element is considered a child node of that element. Consider the following HTML:

```
1 | <div class="container">
2 |   <div class="hello">Hello</div>
3 |   <div class="goodbye">Goodbye</div>
4 | </div>
```

We can target any element for removal:

```
1 | $(' .hello' ).empty();
```

This will result in a DOM structure with the `Hello` text deleted:

```
1 | <div class="container">
2 |   <div class="hello"></div>
3 |   <div class="goodbye">Goodbye</div>
4 | </div>
```

If we had any number of nested elements inside `<div`

`class="hello">`, they would be removed, too.

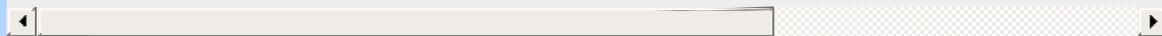
To avoid memory leaks, jQuery removes other constructs such as data and event handlers from the child elements before removing the elements themselves.

If you want to remove elements without destroying their data or event handlers (so they can be re-added later), use `.detach()` instead.

## Example:

*Removes all child nodes (including text nodes) from all paragraphs*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     p { background:yellow; }
6   </style>
7   <script src="http://code.jquery.com/jquery-1.10.2.min.js"></script>
8 </head>
9 <body>
10  <p>
11    Hello, <span>Person</span> <a href="javascript:void(0)">Click here</a>
12  </p>
13
14  <button>Call empty() on above paragraph</button>
15  <script>
16    $("button").click(function () {
17      $("p").empty();
18    });
19  </script>
20
21 </body>
22 </html>
```



Demo

**A new version of this book is available!**

---

# :empty Selector

Categories: [Selectors](#) > [Content Filter](#)

## empty selector

**Description:** *Select all elements that have no children (including text nodes).*

**jQuery( ":empty" )**

version added: [1.0](#)

This is the inverse of `:parent`.

One important thing to note with `:empty` (and `:parent`) is that child elements include text nodes.

The W3C recommends that the `<p>` element have at least one child node, even if that child is merely text (see <http://www.w3.org/TR/html401/struct/text.html#edef-P>). Some other elements, on the other hand, are empty (i.e. have no children) by definition: `<input>`, `<img>`, `<br>`, and `<hr>`, for example.

## Example:

*Find all elements that are empty - they don't have child elements or text.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5
6   td { text-align:center; }
7 </style>
8   <script src="http://code.jquery.com/jquery-
9 </head>
10 <body>
11   <table border="1">
12     <tr><td>TD #0</td><td></td></tr>
13     <tr><td>TD #2</td><td></td></tr>
14
15     <tr><td></td><td>TD#5</td></tr>
16   </table>
17   <script>$("td:empty").text("was empty!").css(
18
19 </body>
20 </html>
```

Demo

**A new version of this book is available!**

---

# :enabled Selector

Categories: [Selectors](#) > [Form](#)

## enabled selector

**Description:** *Selects all elements that are enabled.*

**jQuery( ":enabled" )**

version added: [1.0](#)

As with other pseudo-class selectors (those that begin with a ":") it is recommended to precede it with a tag name or some other selector; otherwise, the universal selector ("\*") is implied. In other words, the bare `$('.enabled')` is equivalent to `$('.*:enabled')`, so `$('.input:enabled')` or similar should be used instead.

Although their resulting selections are usually the same, `:enabled` selector is subtly different from `:not([disabled])`; `:enabled` selects elements that have their boolean `disabled` property strictly equal to `false`, while `:not([disabled])` selects elements that do not have a `disabled` *attribute* set (regardless of its value).

The `:enabled` selector should only be used for selecting HTML elements that support the `disabled` attribute (`<button>`, `<input>`, `<optgroup>`, `<option>`, `<select>`, and `<textarea>`).

## Example:

*Find all input elements that are enabled.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <script src="http://code.jquery.com/jquery-1.9.1.js"></script>
5  </head>
6  <body>
7    <form>
8
9      <input name="email" disabled="disabled" type="text" />
10     <input name="id" type="text" />
11   </form>
12   <script>$("input:enabled").val("this is it");</script>
13
14 </body>
15 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# .end()

Categories: [Traversing](#) > [Miscellaneous Traversing](#)

`.end()`

Returns: `jQuery`

**Description:** *End the most recent filtering operation in the current chain and return the set of matched elements to its previous state.*

`.end()`

version added: `1.0`

This method does not accept any arguments.

Most of jQuery's [DOM traversal](#) methods operate on a jQuery object instance and produce a new one, matching a different set of DOM elements. When this happens, it is as if the new set of elements is pushed onto a stack that is maintained inside the object. Each successive filtering method pushes a new element set onto the stack. If we need an older element set, we can use `end()` to pop the sets back off of the stack.

Suppose we have a couple short lists on a page:

```
1 | <ul class="first">
2 |   <li class="foo">list item 1</li>
3 |   <li>list item 2</li>
4 |   <li class="bar">list item 3</li>
5 | </ul>
6 | <ul class="second">
7 |   <li class="foo">list item 1</li>
8 |   <li>list item 2</li>
9 |   <li class="bar">list item 3</li>
10| </ul>
```

The `end()` method is useful primarily when exploiting jQuery's chaining properties. When not using chaining, we can usually just call up a previous object by variable name, so we don't need to manipulate the stack. With `end()`, though, we can string all the method calls together:

```
1 | $( 'ul.first' ).find( '.foo' ).css( 'background-color', 'red' );
2 |   .end().find( '.bar' ).css( 'background-color', 'green' );
```

This chain searches for items with the class `foo` within the first list only and turns their backgrounds red. Then `end()` returns the object to its state before the call to `find()`, so the second `find()` looks for `.bar` inside `<ul class="first">`, not just inside that list's `<li class="foo">`, and turns the matching elements' backgrounds green. The net result is that items 1 and 3 of the first list have a colored background, and none of the items from the second list do.

A long jQuery chain can be visualized as a structured code block, with filtering methods providing the openings of nested blocks and `end()` methods closing them:

```
1 | $( 'ul.first' ).find( '.foo' )
2 |   .css( 'background-color', 'red' )
3 |   .end().find( '.bar' )
4 |     .css( 'background-color', 'green' )
5 |     .end();
```

The last `end()` is unnecessary, as we are discarding the jQuery object immediately thereafter. However, when the code is written in this form, the `end()` provides visual symmetry and a sense of completion —making the program, at least to the eyes of some developers, more readable, at the cost of a slight hit to performance as it is an additional function call.

## Examples:

**Example:** *Selects all paragraphs, finds span elements inside these and reverts the selection back to the paragraphs.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5      p, div { margin:1px; padding:1px; font-weight:
6              font-size:16px; }
7      div { color:blue; }
8      b { color:red; }
9    </style>
10   <script src="http://code.jquery.com/jquery-1.4.2.js"></script>
11 </head>
12 <body>
13   <p>
14     Hi there <span>how</span> are you <span>doing</span>?
15   </p>
16
17   <p>
18     This <span>span</span> is one of
19     several <span>spans</span> in this
20     <span>sentence</span>.
21   </p>
22
23   <div>
24     Tags in jQuery object initially: <b></b>
25   </div>
26   <div>
27     Tags in jQuery object after find: <b></b>
28
29   </div>
```

```

30 <div>
31     Tags in jQuery object after end: <b></b>
32 </div>
33 <script>
34 jQuery.fn.showTags = function (n) {
35     var tags = this.map(function () {
36         return this.tagName;
37     })
38     .get().join(", ");
39     $("b:eq(" + n + ")").text(tags);
40     return this;
41 };
42
43 $("p").showTags(0)
44     .find("span")
45     .showTags(1)
46     .css("background", "yellow")
47     .end()
48     .showTags(2)
49     .css("font-style", "italic");
50
51 </script>
52
53 </body>
54 </html>

```

## Demo

**Example:** *Selects all paragraphs, finds span elements inside these, and reverts the selection back to the paragraphs.*

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4 <style>p { margin:10px; padding:10px; }</st

```

```
5   <script src="http://code.jquery.com/jquery-latest.js"
6   </head>
7   <body>
8     <p><span>Hello</span>, how are you?</p>
9     <script>$("p").find("span").end().css("border",
10
11   </body>
12   </html>
```

Demo

POWERED BYHERONOTE

A new version of this book is available!

---

# .eq()

Categories: [Traversing](#) > [Filtering](#)

`.eq( index )`

Returns: [jQuery](#)

**Description:** Reduce the set of matched elements to the one at the specified index.

`.eq( index )`

version added: [1.1.2](#)

**index**

Type: [Integer](#)

An integer indicating the 0-based position of the element.

`.eq( -index )`

version added: [1.4](#)

**-index**

Type: [Integer](#)

An integer indicating the position of the element, counting backwards from the last element in the set.

Given a jQuery object that represents a set of DOM elements, the `.eq()` method constructs a new jQuery object from one element within that set. The supplied index identifies the position of this element in the set.

Consider a page with a simple list on it:

```
1 | <ul>
2 |   <li>list item 1</li>
3 |   <li>list item 2</li>
4 |   <li>list item 3</li>
5 |   <li>list item 4</li>
6 |   <li>list item 5</li>
7 | </ul>
```

We can apply this method to the set of list items:

```
1 | $('li').eq(2).css('background-color', 'red');
```

A code editor window with a light blue background. The code is: `1 | $('li').eq(-2).css('background-color', 'red')`. The `li` and `-2` are highlighted in blue, and `background-color` and `red` are highlighted in red. A scrollbar is visible at the bottom.

The result of this call is a red background for item 3. Note that the supplied index is zero-based, and refers to the position of the element within the jQuery object, not within the DOM tree.

Providing a negative number indicates a position starting from the end of the set, rather than the beginning. For example:

A code editor window with a light blue background. The code is: `1 | $('li').eq(-2).css('background-color', 'red')`. The `li` and `-2` are highlighted in blue, and `background-color` and `red` are highlighted in red. A scrollbar is visible at the bottom.

```
1 | $('li').eq(-2).css('background-color', 'red')
```

This time list item 4 is turned red, since it is two from the end of the set.

If an element cannot be found at the specified zero-based index, the method constructs a new jQuery object with an empty set and a `length` property of 0.

A code editor window with a light blue background. The code is: `1 | $('li').eq(5).css('background-color', 'red')`. The `li` and `5` are highlighted in blue, and `background-color` and `red` are highlighted in red. A scrollbar is visible at the bottom.

```
1 | $('li').eq(5).css('background-color', 'red')
```

Here, none of the list items is turned red, since `.eq(5)` indicates the sixth of five list items.

## Example:

*Turn the div with index 2 blue by adding an appropriate class.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     div { width:60px; height:60px; margin:10px;
6           border:2px solid blue; }
7     .blue { background:blue; }
8   </style>
9   <script src="http://code.jquery.com/jquery-
10 </head>
11 <body>
12   <div></div>
13   <div></div>
14   <div></div>
15
16   <div></div>
17   <div></div>
18   <div></div>
19 <script>
20   $("body").find("div").eq(2).addClass("blue");
21 </script>
22
23 </body>
24 </html>
```

Demo

**A new version of this book is available!**

---

# :eq() Selector

Categories: [Selectors](#) > [Basic Filter](#) | [Selectors](#) > [jQuery Extensions](#)

## eq selector

**Description:** *Select the element at index `n` within the matched set.*

**jQuery( ":eq(index)" )**

version added: [1.0](#)

**index:** Zero-based index of the element to match.

**jQuery( ":eq(-index)" )**

version added: [1.8](#)

**-index:** Zero-based index of the element to match, counting backwards from the last element.

The index-related selectors (`:eq()`, `:lt()`, `:gt()`, `:even`, `:odd`) filter the set of elements that have matched the expressions that precede them. They narrow the set down based on the order of the elements within this matched set. For example, if elements are first selected with a class selector (`.myclass`) and four elements are returned, these elements are given indices `0` through `3` for the purposes of these selectors.

Note that since JavaScript arrays use *0-based indexing*, these selectors reflect that fact. This is why `$('.myclass:eq(1)')` selects the second element in the document with the class `myclass`, rather than the first. In contrast, `:nth-child(n)` uses *1-based indexing* to conform to the CSS specification.

Prior to jQuery 1.8, the `:eq(index)` selector did *not* accept a negative value for `index` (though the `.eq(index)` method did).

## Additional Notes:

Because `:eq()` is a jQuery extension and not part of the CSS specification, queries using `:eq()` cannot take advantage of the performance boost provided by the native DOM `querySelectorAll()` method. For better performance in modern browsers, use `$("your-pure-css-selector").eq(index)` instead.



## Examples:

**Example:** *Finds the third td.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
5 </head>
6 <body>
7   <table border="1">
8     <tr><td>TD #0</td><td>TD #1</td><td>TD #2</td><td>TD #3</td><td>TD #4</td><td>TD #5</td><td>TD #6</td><td>TD #7</td><td>TD #8</td></tr>
9     <tr><td>TD #3</td><td>TD #4</td><td>TD #5</td><td>TD #6</td><td>TD #7</td><td>TD #8</td></tr>
10    <tr><td>TD #6</td><td>TD #7</td><td>TD #8</td></tr>
11  </table>
12  <script>$("td:eq(2)").css("color", "red");</script>
13
14 </body>
15 </html>
```

### Demo

**Example:** *Apply three different styles to list items to demonstrate that `.eq()` is designed to select a single element while `.nth-child()` or `.eq()` within a looping construct such as `.each()` can select multiple elements.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
5 </head>
6 <body>
7   <ul class="nav">
```

```

8     <li>List 1, item 1</li>
9     <li>List 1, item 2</li>
10    <li>List 1, item 3</li>
11  </ul>
12  <ul class="nav">
13    <li>List 2, item 1</li>
14    <li>List 2, item 2</li>
15    <li>List 2, item 3</li>
16  </ul>
17
18  <script>
19    /* applies yellow background color to a single <li> */
20    $("ul.nav li:eq(1)").css( "backgroundColor", "#ff0"
21
22    /* applies italics to text of the second <li> within ea
23    $("ul.nav").each(function(index) {
24      $(this).find("li:eq(1)").css( "fontStyle", "italic"
25    });
26
27    /* applies red text color to descendants of <ul class="
28    /* for each <li> that is the second child of its parent
29    $("ul.nav li:nth-child(2)").css( "color", "red"
30  </script>
31
32  </body>
33  </html>

```

## Demo

**Example** • Add a class to List 2, item 2 by targeting the second to last <li>

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4  <style>

```

```
5  .foo {
6    color: blue;
7    background-color: yellow;
8  }
9  </style>
10 <script src="http://code.jquery.com/jquery-latest.js"
11 </head>
12 <body>
13   <ul class="nav">
14     <li>List 1, item 1</li>
15     <li>List 1, item 2</li>
16     <li>List 1, item 3</li>
17   </ul>
18   <ul class="nav">
19     <li>List 2, item 1</li>
20     <li>List 2, item 2</li>
21     <li>List 2, item 3</li>
22   </ul>
23
24   <script>
25     $( "li:eq(-2)" ).addClass( "foo" )
26   </script>
27
28 </body>
29 </html>
```

Demo

**A new version of this book is available!**

---

# .error()

Categories: [Events](#) > [Browser Events](#)

`.error( handler(eventObject) )`

Returns: [jQuery](#)  
version deprecated: [1.8](#)

**Description:** *Bind an event handler to the "error" JavaScript event.*

`.error( handler(eventObject) )`

version added: [1.0](#)

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute when the event is triggered.

`.error( [eventData ],  
handler(eventObject) )`

version added: [1.4.3](#)

**eventData**

Type: [Object](#)

An object containing data that will be passed to the event handler.

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute each time the event is triggered.

This method is a shortcut for `.on('error', handler)`.

The `error` event is sent to elements, such as images, that are referenced by a document and loaded by the browser. It is called if the element was not loaded correctly.

For example, consider a page with a simple image element:

```
1 | <img alt="Book" id="book" />
```

The event handler can be bound to the image:

```
1 | $('#book')
```

```
2 | .error(function() {  
3 |     alert('Handler for .error() called.')
```

```
4 | })
```

```
5 | .attr("src", "missing.png");
```

If the image cannot be loaded (for example, because it is not present at the supplied URL), the alert is displayed:

Handler for .error() called.

*The event handler **must** be attached before the browser fires the error event, which is why the example sets the src attribute after attaching the handler. Also, the error event may not be correctly fired when the page is served locally; `error` relies on HTTP status codes and will generally not be triggered if the URL uses the `file:` protocol.*

Note: A jQuery error event handler should not be attached to the window object. The browser fires the window's error event when a script error occurs. However, the window error event receives different arguments and has different return value requirements than conventional event handlers. Use `window.onerror` instead.

## Example:

*To hide the "broken image" icons for IE users, you can try:*

```
1 | $("img")
2 |   .error(function(){
3 |     $(this).hide();
4 |   })
5 |   .attr("src", "missing.png");
```

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# :even Selector

Categories: [Selectors](#) > [Basic Filter](#) | [Selectors](#) > [jQuery Extensions](#)

## even selector

**Description:** *Selects even elements, zero-indexed. See also [odd](#).*

## jQuery( ":even" )

version added: [1.0](#)

In particular, note that the *0-based indexing* means that, counter-intuitively, `:even` selects the first element, third element, and so on within the matched set.

### Additional Notes:

Because `:even` is a jQuery extension and not part of the CSS specification, queries using `:even` cannot take advantage of the performance boost provided by the native DOM `querySelectorAll()` method. To achieve the best performance when using `:even` to select elements, first select the elements using a pure CSS selector, then use `.filter(":even")`.

Selected elements are in the order of their appearance in the document.

## Example:

*Find even table rows, matching the first, third and so on (index 0, 2, 4 etc.).*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     table {
6       background:#eeeeee;
7     }
8   </style>
9   <script src="http://code.jquery.com/jquery-
10 </head>
11 <body>
12   <table border="1">
13     <tr><td>Row with Index #0</td></tr>
14     <tr><td>Row with Index #1</td></tr>
15
16     <tr><td>Row with Index #2</td></tr>
17     <tr><td>Row with Index #3</td></tr>
18   </table>
19   <script>$("tr:even").css("background-color",
20
21 </body>
22 </html>
```

Demo

**A new version of this book is available!**

---

# event.currentTarget

Categories: [Events](#) > [Event Object](#)

`event.currentTarget`

Returns: [Element](#)

**Description:** *The current DOM element within the event bubbling phase.*

`event.currentTarget`

version added: [1.3](#)

This property will typically be equal to the `this` of the function.

*If you are using [jQuery.proxy](#) or another form of scope manipulation, `this` will be equal to whatever context you have provided, not `event.currentTarget`*

## Example:

*Alert that currentTarget matches the `this` keyword.*

```
1 | $("p").click(function(event) {  
2 |     alert( event.currentTarget === this ); // true  
3 | });
```

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# event.data

Categories: [Events](#) > [Event Object](#)

`event.data`

*Returns:* `Object`

**Description:** *An optional object of data passed to an event method when the current executing handler is bound.*

`event.data`

version added: `1.1`

## Example:

Within a `for` loop, pass the value of `i` to the `.on()` method so that the current iteration's value is preserved.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-1.10.2.min.js"></script>
5 </head>
6 <body>
7
8   <button> 0 </button>
9   <button> 1 </button>
10  <button> 2 </button>
11  <button> 3 </button>
12  <button> 4 </button>
13
14  <div id="log"></div>
15
16  <script>
17    var logDiv = $("#log");
18
19    /* Note: This code is for demonstration purposes only. */
20    for (var i = 0; i < 5; i++) {
21      $("button").eq(i).on("click", {value: i}, function(event) {
22        var msgs = [
23          "button = " + $(this).index(),
24          "event.data.value = " + event.data.value,
25          "i = " + i
26        ];
27        logDiv.append( msgs.join(", ") + "<br>" );
28      });
29    }
```

```
30 </script>
31
32 </body>
33 </html>
```

Demo

POWERED BY **HERONOTE**

A new version of this book is available!

---

# event.delegateTarget

Categories: [Events](#) > [Event Object](#) | [Events](#)

`event.delegateTarget`

Returns: [Element](#)

**Description:** *The element where the currently-called jQuery event handler was attached.*

**event.delegateTarget**

version added: [1.7](#)

This property is most often useful in delegated events attached by `.delegate()` or `.on()`, where the event handler is attached at an ancestor of the element being processed. It can be used, for example, to identify and remove event handlers at the delegation point.

For non-delegated event handlers attached directly to an element, `event.delegateTarget` will always be equal to `event.currentTarget`.

## Example:

*When a button in any box class is clicked, change the box's background color to red.*

```
1 | $(".box").on("click", "button", function(event  
2 |     $(event.delegateTarget).css("background-colo  
3 | });
```

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# event.preventDefault()

Categories: [Events](#) > [Event Object](#)

`event.isDefaultPrevented()`

Returns: [Boolean](#)

**Description:** Returns whether [event.preventDefault\(\)](#) was ever called on this event object.

**event.isDefaultPrevented()**

version added: [1.3](#)

This method does not accept any arguments.

## Example:

*Checks whether `event.preventDefault()` was called.*

```
1 | $("a").click(function(event){
2 |     alert( event.isDefaultPrevented() ); // false
3 |     event.preventDefault();
4 |     alert( event.isDefaultPrevented() ); // true
5 | });
```

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# event.isImmediatePropagationSto

Categories: [Events](#) > [Event Object](#)

`event.isImmediatePropagationStopped()` Returns: [Boolean](#)

**Description:** *Returns whether `event.stopImmediatePropagation()` was ever called on this event object.*

version added: [1.3](#)

**`event.isImmediatePropagationStopped()`**

This method does not accept any arguments.

This property was introduced in [DOM level 3](#).

## Example:

*Checks whether `event.stopImmediatePropagation()` was called.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-1.10.2.min.js"></script>
5 </head>
6 <body>
7
8   <button>click me</button>
9   <div id="stop-log"></div>
10
11 <script>
12
13 function immediatePropStopped(e) {
14   var msg = "";
15   if ( e.isImmediatePropagationStopped() ) {
16     msg = "called"
17   } else {
18     msg = "not called";
19   }
20   $("#stop-log").append( "<div>" + msg + "</div>" );
21 }
22
23 $("button").click(function(event) {
24   immediatePropStopped(event);
25   event.stopImmediatePropagation();
26   immediatePropStopped(event);
27 });
28 </script>
29
30 </body>
```

31 | `</html>`

Demo

POWERED BY **HERONOTE**

A new version of this book is available!

---

# event.isPropagationStopped()

Categories: [Events](#) > [Event Object](#)

`event.isPropagationStopped()`

Returns: [Boolean](#)

**Description:** Returns whether [event.stopPropagation\(\)](#) was ever called on this event object.

**event.isPropagationStopped()**

version added: [1.3](#)

This method does not accept any arguments.

This event method is described in the [W3C DOM Level 3 specification](#).

## Example:

*Checks whether event.stopPropagation() was called*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-1.10.2.min.js"></script>
5 </head>
6 <body>
7
8   <button>click me</button>
9   <div id="stop-log"></div>
10
11 <script>
12
13 function propStopped(e) {
14   var msg = "";
15   if ( e.isPropagationStopped() ) {
16     msg = "called"
17   } else {
18     msg = "not called";
19   }
20   $( "#stop-log" ).append( "<div>" + msg + "</div>" );
21 }
22
23 $( "button" ).click( function( event ) {
24   propStopped( event );
25   event.stopPropagation();
26   propStopped( event );
27 });
28 </script>
29
30 </body>
31 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# event.metaKey

Categories: [Events](#) > [Event Object](#)

`event.metaKey`

Returns: [Boolean](#)

**Description:** *Indicates whether the META key was pressed when the event fired.*

`event.metaKey`

version added: [1.0.4](#)

Returns a boolean value (`true` or `false`) that indicates whether or not the META key was pressed at the time the event fired. This key might map to an alternative key name on some platforms.

## Example:

*Determine whether the META key was pressed when the event fired.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>body {background-color: #eef; }
5     div { padding: 20px; }</style>
6   <script src="http://code.jquery.com/jquery-...
7 </head>
8 <body>
9
10 <button value="Test" name="Test" id="checkMetaKey">Test</button>
11 <div id="display"></div>
12 <script>
13   $('#checkMetaKey').click(function(e){
14     $('#display').text(e.metaKey);
15   });
16 </script>
17
18 </body>
19 </html>
```

Demo

**A new version of this book is available!**

---

# event.namespace

Categories: [Events](#) > [Event Object](#)

`event.namespace`

Returns: `String`

**Description:** *The namespace specified when the event was triggered.*

`event.namespace`

version added: `1.4.3`

This will likely be used primarily by plugin authors who wish to handle tasks differently depending on the event namespace used.

## Example:

*Determine the event namespace used.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <script src="http://code.jquery.com/jquery-
5  </head>
6  <body>
7
8  <button>display event.namespace</button>
9  <p></p>
10
11 <script>
12 $("p").on("test.something", function(event) {
13   alert( event.namespace );
14 });
15 $("button").click(function(event) {
16   $("p").trigger("test.something");
17 });
18 </script>
19
20 </body>
21 </html>
```

Demo

**A new version of this book is available!**

---

# event.pageX

Categories: [Events](#) > [Event Object](#)

`event.pageX`

Returns: [Number](#)

**Description:** *The mouse position relative to the left edge of the document.*

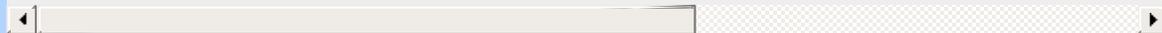
`event.pageX`

version added: [1.0.4](#)

## Example:

*Show the mouse position relative to the left and top edges of the document (within the iframe).*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>body {background-color: #eef; }
5     div { padding: 20px; }</style>
6   <script src="http://code.jquery.com/jquery-...
7 </head>
8 <body>
9   <div id="log"></div>
10  <script>$(document).on('mousemove', function(e
11      $("#log").text("e.pageX: " + e.pageX)
12  }); </script>
13
14 </body>
15 </html>
```



Demo

POWERED BY HERONOTE

A new version of this book is available!

---

# event.pageY

Categories: [Events](#) > [Event Object](#)

`event.pageY`

Returns: [Number](#)

**Description:** *The mouse position relative to the top edge of the document.*

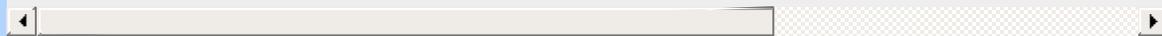
`event.pageY`

version added: [1.0.4](#)

## Example:

*Show the mouse position relative to the left and top edges of the document (within this iframe).*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>body {background-color: #eef; }
5     div { padding: 20px; }
6   </style>
7   <script src="http://code.jquery.com/jquery-
8 </head>
9 <body>
10  <div id="log"></div>
11  <script>
12    $(document).on( 'mousemove', function(e){
13      $("#log").text("e.pageX: " + e.pageX + ", e
14    });
15  </script>
16
17 </body>
18 </html>
```



Demo

**A new version of this book is available!**

---

# event.preventDefault()

Categories: [Events](#) > [Event Object](#)

`event.preventDefault()`

Returns: `undefined`

**Description:** *If this method is called, the default action of the event will not be triggered.*

`event.preventDefault()`

version added: [1.0](#)

This method does not accept any arguments.

For example, clicked anchors will not take the browser to a new URL. We can use `event.isDefaultPrevented()` to determine if this method has been called by an event handler that was triggered by this event.

## Example:

*Cancel the default action (navigation) of the click.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-1.10.2.min.js"></script>
5 </head>
6 <body>
7
8 <a href="http://jquery.com">default click act.
9 <div id="log"></div>
10
11 <script>
12 $( "a" ).click( function( event ) {
13   event.preventDefault();
14   $( '<div/>' )
15     .append( 'default ' + event.type + ' preventDefault()' )
16     .appendTo( '#log' );
17 });
18 </script>
19
20 </body>
21 </html>
```

Demo

**A new version of this book is available!**

---

# event.relatedTarget

Categories: [Events](#) > [Event Object](#)

`event.relatedTarget`

Returns: [Element](#)

**Description:** *The other DOM element involved in the event, if any.*

**event.relatedTarget**

version added: [1.1.4](#)

For `mouseout`, indicates the element being entered; for `mouseover`, indicates the element being exited.

## Example:

*On mouseout of anchors, alert the element type being entered.*

```
1 | $("a").mouseout(function(event) {  
2 |     alert(event.relatedTarget.nodeName); // "DIV"  
3 | });
```

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# event.result

Categories: [Events](#) > [Event Object](#)

`event.result`

Returns: [Object](#)

**Description:** *The last value returned by an event handler that was triggered by this event, unless the value was `undefined`.*

`event.result`

version added: [1.3](#)

This property can be useful for getting previous return values of custom events.

## Example:

*Display previous handler's return value*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-1.10.2.min.js"></script>
5 </head>
6 <body>
7
8 <button>display event.result</button>
9 <p></p>
10
11 <script>
12   $("button").click(function(event) {
13     return "hey";
14   });
15   $("button").click(function(event) {
16     $("p").html( event.result );
17   });
18 </script>
19
20 </body>
21 </html>
```

Demo

**A new version of this book is available!**

---

# event.stopImmediatePropagation()

Categories: [Events](#) > [Event Object](#)

`event.stopImmediatePropagation()`

*Returns:*

**Description:** *Keeps the rest of the handlers from being executed and prevents the event from bubbling up the DOM tree.*

`event.stopImmediatePropagation()`

version added: [1.3](#)

This method does not accept any arguments.

In addition to keeping any additional handlers on an element from being executed, this method also stops the bubbling by implicitly calling `event.stopPropagation()`. To simply prevent the event from bubbling to ancestor elements but allow other event handlers to execute on the same element, we can use `event.stopPropagation()` instead.

Use `event.isImmediatePropagationStopped()` to know whether this method was ever called (on that event object).

## Additional Notes:

Since the `.live()` method handles events once they have propagated to the top of the document, it is not possible to stop propagation of live events. Similarly, events handled by `.delegate()` will propagate to the elements to which they are delegated; event handlers bound on any elements below it in the DOM tree will already have been executed by the time the delegated event handler is called. These handlers, therefore, may prevent the delegated handler from triggering by calling `event.stopPropagation()` or returning `false`.

## Example:

*Prevents other event handlers from being called.*

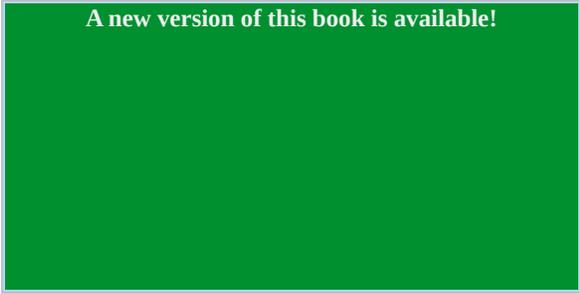
```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5   p { height: 30px; width: 150px; background-color: #f00; }
6   div { height: 30px; width: 150px; background-color: #f00; }
7 </style>
8   <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
9 </head>
10 <body>
11   <p>paragraph</p>
12   <div>division</div>
13   <script>
14     $("p").click(function(event){
15       event.stopImmediatePropagation();
16     });
17     $("p").click(function(event){
18       // This function won't be executed
19       $(this).css("background-color", "#f00");
20     });
21     $("div").click(function(event) {
22       // This function will be executed
23       $(this).css("background-color", "#f00");
24     });</script>
25
26 </body>
27 </html>
```

Demo

---

POWERED BY **HERONOTE**

A new version of this book is available!



---

# event.stopPropagation()

Categories: [Events](#) > [Event Object](#)

`event.stopPropagation()`

Returns:

**Description:** *Prevents the event from bubbling up the DOM tree, preventing any parent handlers from being notified of the event.*

`event.stopPropagation()`

version added: [1.0](#)

This method does not accept any arguments.

We can use `event.isPropagationStopped()` to determine if this method was ever called (on that event object).

This method works for custom events triggered with `trigger()`, as well.

Note that this will not prevent other handlers *on the same element* from running.

## Additional Notes:

Since the `.live()` method handles events once they have propagated to the top of the document, it is not possible to stop propagation of live events. Similarly, events handled by `.delegate()` will propagate to the elements to which they are delegated; event handlers bound on any elements below it in the DOM tree will already have been executed by the time the delegated event handler is called. These handlers, therefore, may prevent the delegated handler from triggering by calling `event.stopPropagation()` or returning `false`.

## Example:

*Kill the bubbling on the click event.*

```
1 | $("p").click(function(event){  
2 |     event.stopPropagation();  
3 |     // do something  
4 | });
```

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# event.target

Categories: [Events](#) > [Event Object](#)

`event.target`

Returns: [Element](#)

**Description:** *The DOM element that initiated the event.*

**event.target**

version added: [1.0](#)

The `target` property can be the element that registered for the event or a descendant of it. It is often useful to compare `event.target` to `this` in order to determine if the event is being handled due to event bubbling. This property is very useful in event delegation, when events bubble.

## Examples:

**Example:** *Display the tag's name on click*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5    span, strong, p {
6      padding: 8px; display: block; border: 1px solid #ccc;
7    }
8    <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
9  </head>
10 <body>
11
12 <div id="log"></div>
13 <div>
14   <p>
15     <strong><span>click</span></strong>
16   </p>
17 </div>
18 <script>$("body").click(function(event) {
19   $("#log").html("clicked: " + event.target.tagName);
20 }); </script>
21
22 </body>
23 </html>
```

### Demo

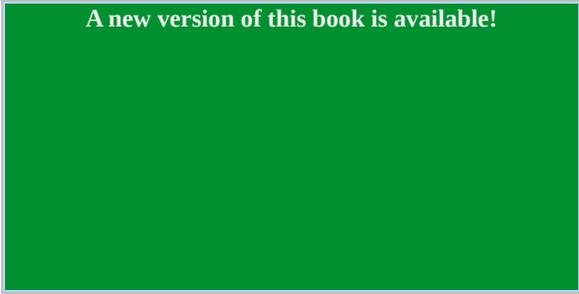
**Example:** *Implements a simple event delegation: The click handler is added to an unordered list and the children of its li children are hidden. Clicking one of the li children toggles (see toggle()) their children.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-latest.js">
5 </head>
6 <body>
7
8 <ul>
9   <li>item 1
10     <ul>
11       <li>sub item 1-a</li>
12       <li>sub item 1-b</li>
13     </ul>
14   </li>
15   <li>item 2
16     <ul>
17       <li>sub item 2-a</li>
18       <li>sub item 2-b</li>
19     </ul>
20   </li>
21 </ul>
22 <script>function handler(event) {
23   var $target = $(event.target);
24   if( $target.is("li") ) {
25     $target.children().toggle();
26   }
27 }
28 $("ul").click(handler).find("ul").hide();</script>
29
30 </body>
31 </html>
```

Demo

POWERED BY **HERONOTE**

A new version of this book is available!



---

# event.timeStamp

Categories: [Events](#) > [Event Object](#)

`event.timeStamp`

Returns: [Number](#)

**Description:** *The difference in milliseconds between the time the browser created the event and January 1, 1970.*

`event.timeStamp`

version added: [1.2.6](#)

This property can be useful for profiling event performance by getting the `event.timeStamp` value at two points in the code and noting the difference. To simply determine the current time inside an event handler, use `(new Date).getTime()` instead.

Note: Due to a [bug open since 2004](#), this value is not populated correctly in Firefox and it is not possible to know the time the event was created in that browser.

## Example:

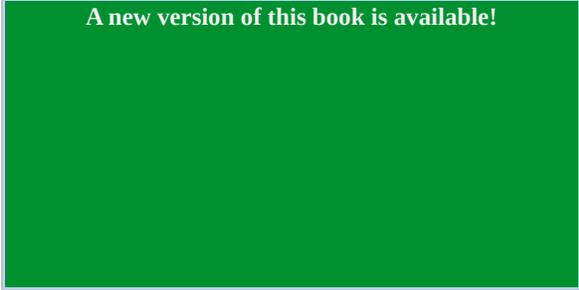
*Display the time since the click handler last executed.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5   div { height: 100px; width: 300px; margin: 10px;
6         background-color: #ffd; overflow: auto;
7   }
8   <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
9 </head>
10 <body>
11
12 <div>Click.</div>
13 <script>
14 var last, diff;
15 $('div').click(function(event) {
16   if ( last ) {
17     diff = event.timeStamp - last
18     $('div').append('time since last event: ' + diff);
19   } else {
20     $('div').append('Click again.<br/>');
21   }
22   last = event.timeStamp;
23 });
24 </script>
25
26 </body>
27 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

# event.type

Categories: [Events](#) > [Event Object](#)

`event.type`

Returns: `String`

**Description:** *Describes the nature of the event.*

`event.type`

version added: `1.0`

## Example:

*On all anchor clicks, alert the event type.*

```
1 | $("a").click(function(event) {  
2 |     alert(event.type); // "click"  
3 | });
```

POWERED BY **HERONOTE**

A new version of this book is available!

---

# event.which

Categories: [Events](#) > [Event Object](#)

`event.which`

Returns: [Number](#)

**Description:** *For key or mouse events, this property indicates the specific key or button that was pressed.*

`event.which`

version added: [1.1.3](#)

The `event.which` property normalizes `event.keyCode` and `event.charCode`. It is recommended to watch `event.which` for keyboard key input. For more detail, read about [event.charCode on the MDC](#).

`event.which` also normalizes button presses (`mousedown` and `mouseup` events), reporting `1` for left button, `2` for middle, and `3` for right. Use `event.which` instead of `event.button`.

## Examples:

**Example:** *Log which key was depressed.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
5 </head>
6 <body>
7
8   <input id="whichkey" value="type something">
9   <div id="log"></div>
10  <script>$('#whichkey').on('keydown', function(e) {
11    $('#log').html(e.type + ': ' + e.which);
12  }); </script>
13
14 </body>
15 </html>
```

Demo

**Example:** *Log which mouse button was depressed.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
5 </head>
6 <body>
7
8   <input id="whichkey" value="type something">
9   <div id="log"></div>
10  <script>
```

```
11 $( '#whichkey' ).on( 'mousedown', function(e){
12     $( '#log' ).html(e.type + ': ' + e.which );
13 });
14 </script>
15
16 </body>
17 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# .fadeIn()

Categories: [Effects](#) > [Fading](#)

`.fadeIn( [duration ] [, complete ] )`

Returns: [jQuery](#)

**Description:** *Display the matched elements by fading them to opaque.*

`.fadeIn( [duration ] [, complete ] )`

version added: [1.0](#)

**duration** (default: `400`)

Type: [Number](#) or [String](#)

A string or number determining how long the animation will run.

**complete**

Type: [Function\(\)](#)

A function to call once the animation is complete.

`.fadeIn( options )`

version added: [1.0](#)

**options**

Type: [PlainObject](#)

A map of additional options to pass to the method.

**duration** (default: `400`)

Type: [Number](#) or [String](#)

A string or number determining how long the animation will run.

**easing** (default: `swing`)

Type: [String](#)

A string indicating which easing function to use for the transition.

**queue** (default: `true`)

Type: [Boolean](#)

A Boolean indicating whether to place the animation in the effects queue. If false, the animation will begin immediately.

**As of jQuery 1.7**, the queue option can also accept a string, in which case the animation is added to the queue represented by that string.

**specialEasing**

Type: [PlainObject](#)

A map of one or more of the CSS properties defined by the properties argument and their corresponding easing functions. **(version added: 1.4)**

### step

Type: [Function](#)( [Number](#) now, [Tween](#) tween )

A function to be called for each animated property of each animated element. This function provides an opportunity to modify the Tween object to change the value of the property before it is set.

### progress

Type: [Function](#)( [Promise](#) animation, [Number](#) progress, [Number](#) remainingMs )

A function to be called after each step of the animation, only once per animated element regardless of the number of animated properties. **(version added: 1.8)**

### complete

Type: [Function](#)()

A function to call once the animation is complete.

### done

Type: [Function](#)( [Promise](#) animation, [Boolean](#) jumpedToEnd )

A function to be called when the animation completes (its Promise object is resolved). **(version added: 1.8)**

### fail

Type: [Function](#)( [Promise](#) animation, [Boolean](#) jumpedToEnd )

A function to be called when the animation fails to complete (its Promise object is rejected). **(version added: 1.8)**

### always

Type: [Function](#)( [Promise](#) animation, [Boolean](#) jumpedToEnd )

A function to be called when the animation completes or stops without completing (its Promise object is either resolved or rejected). **(version added: 1.8)**

**.fadeIn( [duration ] [, easing ] [, complete ] )**

version added: [1.4.3](#)

**duration** (default: 400)

Type: [Number](#) or [String](#)

A string or number determining how long the animation will run.

**easing** (default: swing)

Type: [String](#)

A string indicating which easing function to use for the transition.

**complete**

Type: [Function\(\)](#)

A function to call once the animation is complete.

The `.fadeIn()` method animates the opacity of the matched elements.

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings `'fast'` and `'slow'` can be supplied to indicate durations of 200 and 600 milliseconds, respectively. If any other string is supplied, or if the `duration` parameter is omitted, the default duration of 400 milliseconds is used.

We can animate any element, such as a simple image:

```
1 <div id="clickme">
2     Click here
3 </div>
4 
2  <html>
3  <head>
4    <style>
5  span { color:red; cursor:pointer; }
6  div { margin:3px; width:80px; display:none;
7    height:80px; float:left; }
8    div#one { background:#f00; }
9    div#two { background:#0f0; }
10   div#three { background:#00f; }
11 </style>
12   <script src="http://code.jquery.com/jquery-
13 </head>
14 <body>
15   <span>Click here...</span>
16
17     <div id="one"></div>
18     <div id="two"></div>
19     <div id="three"></div>
20 <script>
21 $(document.body).click(function () {
22   $("div:hidden:first").fadeIn("slow");
23 });
24 </script>
25
26 </body>
27 </html>
```

## Demo

**Example:** *Fades a red block in over the text. Once the animation is done, it quickly fades in more text on top.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5      p { position:relative; width:400px; height:
6        div { position:absolute; width:400px; height:
7          font-size:36px; text-align:center;
8          color:yellow; background:red;
9          padding-top:25px;
10         top:0; left:0; display:none; }
11        span { display:none; }
12    </style>
13    <script src="http://code.jquery.com/jquery-
14  </head>
15  <body>
16    <p>
17      Let it be known that the party of the f
18      and the party of the second part are he
19      and hereto directed to assess the alleg
20      for factual correctness... (<a href="#">
21      <div><span>CENSORED!</span></div>
22
23    </p>
24    <script>
25    $("a").click(function () {
26      $("div").fadeIn(3000, function () {
27        $("span").fadeIn(100);
28      });
29      return false;
30    });
31  </script>
32
```

```
33 | </body>  
34 | </html>
```

Demo

POWERED BY **HERONOTE**

A new version of this book is available!

---

# .fadeOut()

Categories: [Effects](#) > [Fading](#)

`.fadeOut( [duration ] [, complete ] )`

Returns: [jQuery](#)

**Description:** *Hide the matched elements by fading them to transparent.*

`.fadeOut( [duration ] [, complete ] )`

version added: [1.0](#)

**duration** (default: `400`)

Type: [Number](#) or [String](#)

A string or number determining how long the animation will run.

**complete**

Type: [Function\(\)](#)

A function to call once the animation is complete.

`.fadeOut( options )`

version added: [1.0](#)

**options**

Type: [PlainObject](#)

A map of additional options to pass to the method.

**duration** (default: `400`)

Type: [Number](#) or [String](#)

A string or number determining how long the animation will run.

**easing** (default: `swing`)

Type: [String](#)

A string indicating which easing function to use for the transition.

**queue** (default: `true`)

Type: [Boolean](#)

A Boolean indicating whether to place the animation in the effects queue. If false, the animation will begin immediately.

**As of jQuery 1.7**, the queue option can also accept a string, in which case the animation is added to the queue represented by that string.

**specialEasing**

Type: [PlainObject](#)

A map of one or more of the CSS properties defined by the properties argument and their corresponding easing functions. **(version added: 1.4)**

### step

Type: [Function](#)( [Number](#) now, [Tween](#) tween )

A function to be called for each animated property of each animated element. This function provides an opportunity to modify the Tween object to change the value of the property before it is set.

### progress

Type: [Function](#)( [Promise](#) animation, [Number](#) progress, [Number](#) remainingMs )

A function to be called after each step of the animation, only once per animated element regardless of the number of animated properties. **(version added: 1.8)**

### complete

Type: [Function](#)()

A function to call once the animation is complete.

### done

Type: [Function](#)( [Promise](#) animation, [Boolean](#) jumpedToEnd )

A function to be called when the animation completes (its Promise object is resolved). **(version added: 1.8)**

### fail

Type: [Function](#)( [Promise](#) animation, [Boolean](#) jumpedToEnd )

A function to be called when the animation fails to complete (its Promise object is rejected). **(version added: 1.8)**

### always

Type: [Function](#)( [Promise](#) animation, [Boolean](#) jumpedToEnd )

A function to be called when the animation completes or stops without completing (its Promise object is either resolved or rejected). **(version added: 1.8)**

**.fadeOut( [duration ] [, easing ] [, complete ] )**

version added: [1.4.3](#)

**duration** (default: `400`)

Type: `Number` or `String`

A string or number determining how long the animation will run.

**easing** (default: `swing`)

Type: `String`

A string indicating which easing function to use for the transition.

**complete**

Type: `Function()`

A function to call once the animation is complete.

The `.fadeOut()` method animates the opacity of the matched elements. Once the opacity reaches 0, the `display` style property is set to `none`, so the element no longer affects the layout of the page.

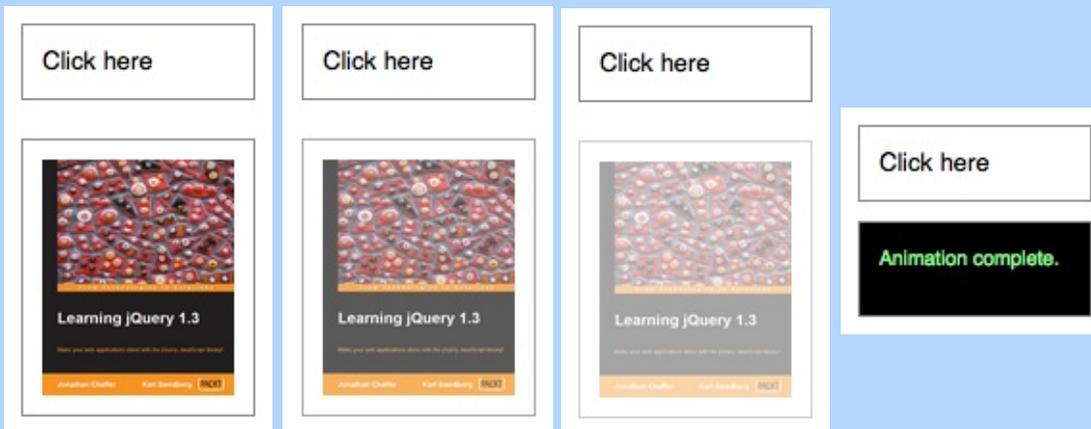
Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings `'fast'` and `'slow'` can be supplied to indicate durations of `200` and `600` milliseconds, respectively. If any other string is supplied, or if the `duration` parameter is omitted, the default duration of `400` milliseconds is used.

We can animate any element, such as a simple image:

```
1 <div id="clickme">
2   Click here
3 </div>
4 
```

With the element initially shown, we can hide it slowly:

```
1 $('#clickme').click(function() {
2   $('#book').fadeOut('slow', function() {
3     // Animation complete.
4   });
});
```



**Note:** To avoid unnecessary DOM manipulation, `.fadeOut()` will not hide an element that is already considered hidden. For information on which elements jQuery considers hidden, see [:hidden Selector](#).

## Easing

As of jQuery 1.4.3, an optional string naming an easing function may be used. Easing functions specify the speed at which the animation progresses at different points within the animation. The only easing implementations in the jQuery library are the default, called `swing`, and one that progresses at a constant pace, called `linear`. More easing functions are available with the use of plug-ins, most notably the [jQuery UI suite](#).

## Callback Function

If supplied, the callback is fired once the animation is complete. This can be useful for stringing different animations together in sequence. The callback is not sent any arguments, but `this` is set to the DOM element being animated. If multiple elements are animated, it is important to note that the callback is executed once per matched

element, not once for the animation as a whole.

**As of jQuery 1.6**, the `.promise()` method can be used in conjunction with the `deferred.done()` method to execute a single callback for the animation as a whole when *all* matching elements have completed their animations ( See the [example for .promise\(\)](#) ).

## Additional Notes:

All jQuery effects, including `.fadeOut()`, can be turned off globally by setting `jQuery.fx.off = true`, which effectively sets the duration to 0. For more information, see [jQuery.fx.off](#).

## Examples:

**Example:** *Animates all paragraphs to fade out completing the animation within 600 milliseconds.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     p { font-size:150%; cursor:pointer; }
6   </style>
7   <script src="http://code.jquery.com/jquery-...
8 </head>
9 <body>
10  <p>
11    If you click on this paragraph
12    you'll see it just fade away.
13  </p>
14  <script>
15    $("p").click(function () {
16      $("p").fadeOut("slow");
17    });
18  </script>
19
20 </body>
21 </html>
```

Demo

**Example:** *Fades out spans in one section that you click on.*

```
1 <!DOCTYPE html>
2 <html>
```

```
3 <head>
4   <style>
5     span { cursor:pointer; }
6     span.hilite { background:yellow; }
7     div { display:inline; color:red; }
8   </style>
9   <script src="http://code.jquery.com/jquery-latest.js">
10 </head>
11 <body>
12   <h3>Find the modifiers - <div></div></h3>
13   <p>
14     If you <span>really</span> want to go outside
15     <span>in the cold</span> then make sure to wear
16     your <span>warm</span> jacket given to you by
17     your <span>favorite</span> teacher.
18   </p>
19   <script>
20
21     $("span").click(function () {
22       $(this).fadeOut(1000, function () {
23         $("div").text("'" + $(this).text() + "' has faded!");
24         $(this).remove();
25       });
26     });
27     $("span").hover(function () {
28       $(this).addClass("hilite");
29     }, function () {
30       $(this).removeClass("hilite");
31     });
32
33   </script>
34
35 </body>
36 </html>
```

**Example:** *Fades out two divs, one with a "linear" easing and one with the default, "swing," easing.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5   .box,
6   button { float:left; margin:5px 10px 5px 0; }
7   .box { height:80px; width:80px; background:#000; }
8   #log { clear:left; }
9
10  </style>
11  <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
12 </head>
13 <body>
14
15 <button id="btn1">fade out</button>
16 <button id="btn2">show</button>
17
18 <div id="log"></div>
19
20 <div id="box1" class="box">linear</div>
21 <div id="box2" class="box">swing</div>
22
23 <script>
24 $( "#btn1" ).click( function() {
25   function complete() {
26     $( "<div/>" ).text( this.id ).appendTo( "#log" );
27   }
28
29   $( "#box1" ).fadeOut( 1600, "linear", complete );
30   $( "#box2" ).fadeOut( 1600, complete );
31 });
32
33 $( "#btn2" ).click( function() {
```

```
34     $("div").show();
35     $("#log").empty();
36 });
37
38 </script>
39
40 </body>
41 </html>
```

Demo

POWERED BY HERONOTE

A new version of this book is available!

---

# **.fadeTo()**

Categories: [Effects](#) > [Fading](#)

`.fadeTo( duration, opacity [, complete ] )`

Returns: [jQuery](#)

**Description:** *Adjust the opacity of the matched elements.*

`.fadeTo( duration, opacity [, complete ] )` version added: [1.0](#)

**duration**

Type: [String](#) or [Number](#)

A string or number determining how long the animation will run.

**opacity**

Type: [Number](#)

A number between 0 and 1 denoting the target opacity.

**complete**

Type: [Function\(\)](#)

A function to call once the animation is complete.

`.fadeTo( duration, opacity [, easing ] [, complete ] )` version added: [1.4.3](#)

**duration**

Type: [String](#) or [Number](#)

A string or number determining how long the animation will run.

**opacity**

Type: [Number](#)

A number between 0 and 1 denoting the target opacity.

**easing**

Type: [String](#)

A string indicating which easing function to use for the transition.

**complete**

Type: [Function\(\)](#)

A function to call once the animation is complete.

The `.fadeTo()` method animates the opacity of the matched elements.

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings `'fast'` and `'slow'` can be supplied to indicate durations of `200` and `600` milliseconds, respectively. If any other string is supplied, the default duration of `400` milliseconds is used. Unlike the other effect methods, `.fadeTo()` requires that `duration` be explicitly specified.

If supplied, the callback is fired once the animation is complete. This can be useful for stringing different animations together in sequence. The callback is not sent any arguments, but `this` is set to the DOM element being animated. If multiple elements are animated, it is important to note that the callback is executed once per matched element, not once for the animation as a whole.

We can animate any element, such as a simple image:

```
1 <div id="clickme">
2   Click here
3 </div>
4 
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-...
5 </head>
6 <body>
7   <p>
8   Click this paragraph to see it fade.
9 </p>
10
11  <p>
12  Compare to this one that won't fade.
13 </p>
14  <script>
15  $("p:first").click(function () {
16  $(this).fadeTo("slow", 0.33);
17  });
18 </script>
19
20 </body>
21 </html>
```

### Demo

**Example:** *Fade div to a random opacity on each click, completing the animation within 200 milliseconds.*

```
1 <!DOCTYPE html>
```

```

2 <html>
3 <head>
4   <style>
5   p { width:80px; margin:0; padding:5px; }
6   div { width:40px; height:40px; position:absolute;
7   div#one { top:0; left:0; background:#f00; }
8   div#two { top:20px; left:20px; background:#0f0;
9   div#three { top:40px; left:40px; background:#00f
10 </style>
11   <script src="http://code.jquery.com/jquery-latest.js"
12 </head>
13 <body>
14   <p>And this is the library that John built...
15
16   <div id="one"></div>
17   <div id="two"></div>
18   <div id="three"></div>
19   <script>
20   $("div").click(function () {
21   $(this).fadeTo("fast", Math.random());
22   });
23 </script>
24
25 </body>
26 </html>

```

## Demo

**Example:** Find the right answer! The fade will take 250 milliseconds and change various styles when it completes.

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>

```

```

5  div, p { width:80px; height:40px; top:0; margin
6  position:absolute; padding-top:8px; }
7  p { background:#fcc; text-align:center; }
8  div { background:blue; }
9  </style>
10 <script src="http://code.jquery.com/jquery-latest.js"
11 </head>
12 <body>
13   <p>Wrong</p>
14 <div></div>
15 <p>Wrong</p>
16 <div></div>
17
18 <p>Right!</p>
19 <div></div>
20 <script>
21 var getPos = function (n) {
22   return (Math.floor(n) * 90) + "px";
23 };
24 $("p").each(function (n) {
25   var r = Math.floor(Math.random() * 3);
26   var tmp = $(this).text();
27   $(this).text($("#p:eq(" + r + ")").text());
28   $("#p:eq(" + r + ")").text(tmp);
29   $(this).css("left", getPos(n));
30 });
31 $("div").each(function (n) {
32   $(this).css("left", getPos(n));
33 })
34 .css("cursor", "pointer")
35 .click(function () {
36   $(this).fadeTo(250, 0.25, function () {
37     $(this).css("cursor", "")
38     .prev().css({"font-weight": "bolder",
39                 "font-style": "italic"});
40   });
41 });

```

```
42  
43 </script>  
44  
45 </body>  
46 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# .fadeToggle()

Categories: [Effects](#) | [Effects](#) > [Fading](#)

`.fadeToggle( [duration ] [, easing ] [, complete ] )`

Returns: [jQuery](#)

**Description:** *Display or hide the matched elements by animating their opacity.*

`.fadeToggle( [duration ] [, easing ] [, complete ] )` version added: [1.4.4](#)

**duration** (default: `400`)

Type: [Number](#) or [String](#)

A string or number determining how long the animation will run.

**easing** (default: `swing`)

Type: [String](#)

A string indicating which easing function to use for the transition.

**complete**

Type: [Function\(\)](#)

A function to call once the animation is complete.

`.fadeToggle( options )`

version added: [1.4.4](#)

**options**

Type: [PlainObject](#)

A map of additional options to pass to the method.

**duration** (default: `400`)

Type: [Number](#) or [String](#)

A string or number determining how long the animation will run.

**easing** (default: `swing`)

Type: [String](#)

A string indicating which easing function to use for the transition.

**queue** (default: `true`)

Type: [Boolean](#)

A Boolean indicating whether to place the animation in the

effects queue. If false, the animation will begin immediately. **As of jQuery 1.7**, the queue option can also accept a string, in which case the animation is added to the queue represented by that string.

---

### **specialEasing**

Type: [PlainObject](#)

A map of one or more of the CSS properties defined by the properties argument and their corresponding easing functions. **(version added: [1.4](#))**

---

### **step**

Type: [Function](#)( [Number](#) now, [Tween](#) tween )

A function to be called for each animated property of each animated element. This function provides an opportunity to modify the Tween object to change the value of the property before it is set.

---

### **progress**

Type: [Function](#)( [Promise](#) animation, [Number](#) progress, [Number](#) remainingMs )

A function to be called after each step of the animation, only once per animated element regardless of the number of animated properties. **(version added: [1.8](#))**

---

### **complete**

Type: [Function](#)()

A function to call once the animation is complete.

---

### **done**

Type: [Function](#)( [Promise](#) animation, [Boolean](#) jumpedToEnd )

A function to be called when the animation completes (its Promise object is resolved). **(version added: [1.8](#))**

---

### **fail**

Type: [Function](#)( [Promise](#) animation, [Boolean](#) jumpedToEnd )

A function to be called when the animation fails to complete (its Promise object is rejected). **(version added: [1.8](#))**

---

### **always**

Type: [Function](#)( [Promise](#) animation, [Boolean](#) jumpedToEnd )

A function to be called when the animation completes or stops without completing (its Promise object is either resolved or rejected). **(version added: [1.8](#))**

The `.fadeToggle()` method animates the opacity of the matched elements. When called on a visible element, the element's `display` style property is set to `none` once the opacity reaches 0, so the element no longer affects the layout of the page.

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings `'fast'` and `'slow'` can be supplied to indicate durations of `200` and `600` milliseconds, respectively.

## Easing

The string representing an easing function specifies the speed at which the animation progresses at different points within the animation. The only easing implementations in the jQuery library are the default, called `swing`, and one that progresses at a constant pace, called `linear`. More easing functions are available with the use of plug-ins, most notably the [jQuery UI suite](#).

## Callback Function

If supplied, the callback is fired once the animation is complete. This can be useful for stringing different animations together in sequence. The callback is not sent any arguments, but `this` is set to the DOM element being animated. If multiple elements are animated, it is important to note that the callback is executed once per matched element, not once for the animation as a whole.

**As of jQuery 1.6**, the `.promise()` method can be used in conjunction with the `deferred.done()` method to execute a single callback for the animation as a whole when *all* matching elements have completed their animations ( See the [example for .promise\(\)](#) ).

## Additional Notes:

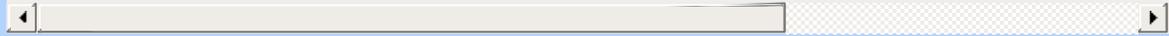
All jQuery effects, including `.fadeToggle()`, can be turned off globally by setting `jQuery.fx.off = true`, which effectively sets

the duration to 0. For more information, see [jQuery.fx.off](http://jquery.fx.off).

## Example:

*Fades first paragraph in or out completing the animation within 600 milliseconds and using a linear easing. Fades last paragraph in or out for 200 milliseconds inserting a "finished" message upon completion.*

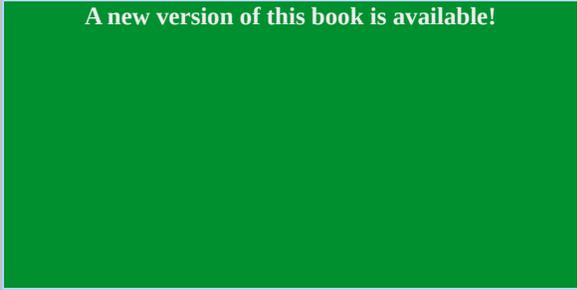
```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-
5 </head>
6 <body>
7
8 <button>fadeToggle p1</button>
9 <button>fadeToggle p2</button>
10 <p>This paragraph has a slow, linear fade.</p>
11
12 <p>This paragraph has a fast animation.</p>
13 <div id="log"></div>
14
15 <script>
16 $("button:first").click(function() {
17   $("p:first").fadeToggle("slow", "linear");
18 });
19 $("button:last").click(function () {
20   $("p:last").fadeToggle("fast", function ()
21     $("#log").append("<div>finished</div>");
22   });
23 });
24 </script>
25
26 </body>
27 </html>
```



Demo

POWERED BY **HERONOTE**

A new version of this book is available!



---

# :file Selector

Categories: [Selectors](#) > [Form](#) | [Selectors](#) > [jQuery Extensions](#)

## file selector

**Description:** *Selects all elements of type file.*

**jQuery( ":file" )**

version added: [1.0](#)

`:file` is equivalent to `[type="file"]`. As with other pseudo-class selectors (those that begin with a ":") it is recommended to precede it with a tag name or some other selector; otherwise, the universal selector ("\*") is implied. In other words, the bare `$(':file')` is equivalent to `$('*:file')`, so `$('input:file')` should be used instead.

### Additional Notes:

Because `:file` is a jQuery extension and not part of the CSS specification, queries using `:file` cannot take advantage of the performance boost provided by the native DOM `querySelectorAll()` method. For better performance in modern browsers, use `[type="file"]` instead.

## Example:

*Finds all file inputs.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5      textarea { height:45px; }
6    </style>
7    <script src="http://code.jquery.com/jquery-
8  </head>
9  <body>
10   <form>
11     <input type="button" value="Input Button"
12     <input type="checkbox" />
13
14     <input type="file" />
15     <input type="hidden" />
16     <input type="image" />
17
18     <input type="password" />
19     <input type="radio" />
20     <input type="reset" />
21
22     <input type="submit" />
23     <input type="text" />
24     <select><option>Option</option></select>
25
26     <textarea></textarea>
27     <button>Button</button>
28   </form>
29   <div>
30   </div>
31 </script>
```

```
32
33     var input = $("input:file").css({background:
34     $("div").text("For this type jQuery found ")
35     .css("color", "red");
36     $("form").submit(function () { return false
37
38 </script>
39
40 </body>
41 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# .filter()

Categories: [Traversing](#) > [Filtering](#)

`.filter( selector )`

Returns: [jQuery](#)

**Description:** Reduce the set of matched elements to those that match the selector or pass the function's test.

**.filter( selector )**

version added: [1.0](#)

**selector**

Type: [Selector](#)

A string containing a selector expression to match the current set of elements against.

**.filter( function(index) )**

version added: [1.0](#)

**function(index)**

Type: [Function\(\)](#)

A function used as a test for each element in the set. `this` is the current DOM element.

**.filter( element )**

version added: [1.4](#)

**element**

Type: [Element](#)

An element to match the current set of elements against.

**.filter( jQuery object )**

version added: [1.4](#)

**jQuery object**

Type: [Object](#)

An existing jQuery object to match the current set of elements against.

Given a jQuery object that represents a set of DOM elements, the `.filter()` method constructs a new jQuery object from a subset of the matching elements. The supplied selector is tested against each element; all elements matching the selector will be included in the result.

Consider a page with a simple list on it:

```

1 | <ul>
2 |   <li>list item 1</li>
3 |   <li>list item 2</li>
4 |   <li>list item 3</li>
5 |   <li>list item 4</li>
6 |   <li>list item 5</li>
7 |   <li>list item 6</li>
8 | </ul>

```

We can apply this method to the set of list items:

```

1 | $('li').filter(':even').css('background-color', 'red');

```

The result of this call is a red background for items 1, 3, and 5, as they match the selector (recall that `:even` and `:odd` use 0-based indexing).

### Using a Filter Function

The second form of this method allows us to filter elements against a function rather than a selector. For each element, if the function returns `true` (or a "truthy" value), the element will be included in the filtered set; otherwise, it will be excluded. Suppose we have a somewhat more involved HTML snippet:

```

1 | <ul>
2 |   <li><strong>list</strong> item 1 -
3 |     one strong tag</li>
4 |   <li><strong>list</strong> item <strong>2</strong>
5 |     two <span>strong tags</span></li>
6 |   <li>list item 3</li>
7 |   <li>list item 4</li>
8 |   <li>list item 5</li>
9 |   <li>list item 6</li>
10| </ul>

```

We can select the list items, then filter them based on their contents:

```
1 | $('li').filter(function(index) {  
2 |     return $('strong', this).length == 1;  
3 | }).css('background-color', 'red');
```

This code will alter the first list item only, as it contains exactly one `<strong>` tag. Within the filter function, `this` refers to each DOM element in turn. The parameter passed to the function tells us the index of that DOM element within the set matched by the jQuery object.

We can also take advantage of the `index` passed through the function, which indicates the 0-based position of the element within the unfiltered set of matched elements:

```
1 | $('li').filter(function(index) {  
2 |     return index % 3 == 2;  
3 | }).css('background-color', 'red');
```

This alteration to the code will cause the third and sixth list items to be highlighted, as it uses the modulus operator (%) to select every item with an `index` value that, when divided by 3, has a remainder of 2.

## Examples:

**Example:** Change the color of all divs: then add a border to those with a "middle" class.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     div { width:60px; height:60px; margin:5px;
6           border:2px white solid;}
7   </style>
8   <script src="http://code.jquery.com/jquery-
9 </head>
10 <body>
11   <div></div>
12
13   <div class="middle"></div>
14   <div class="middle"></div>
15   <div class="middle"></div>
16   <div class="middle"></div>
17
18   <div></div>
19 <script>
20
21   $("div").css("background", "#c8ebcc")
22   .filter(".middle")
23   .css("border-color", "red");
24 </script>
25
26 </body>
27 </html>
```

## Demo

**Example:** *Change the color of all divs: then add a border to the second one (index == 1) and the div with an id of "fourth."*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5      div { width:60px; height:60px; margin:5px;
6            border:3px white solid; }
7    </style>
8    <script src="http://code.jquery.com/jquery-
9  </head>
10 <body>
11
12   <div id="first"></div>
13   <div id="second"></div>
14   <div id="third"></div>
15
16   <div id="fourth"></div>
17   <div id="fifth"></div>
18   <div id="sixth"></div>
19 <script>
20   $("div").css("background", "#b4b0da")
21     .filter(function (index) {
22       return index == 1 || $(this
23         })
24     .css("border", "3px double red");
25
26 </script>
27
28 </body>
29 </html>
```

## Demo

**Example:** Select all divs and filter the selection with a DOM element, keeping only the one with an id of "unique".

```
1 | $("div").filter( document.getElementById("unique") )
```

**Example:** Select all divs and filter the selection with a jQuery object, keeping only the one with an id of "unique".

```
1 | $("div").filter( $("#unique") )
```

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# .find()

Categories: [Traversing](#) > [Tree Traversal](#)

`.find( selector )`

Returns: `jQuery`

**Description:** *Get the descendants of each element in the current set of matched elements, filtered by a selector, jQuery object, or element.*

**.find( selector )**

version added: [1.0](#)

**selector**

Type: [Selector](#)

A string containing a selector expression to match elements against.

**.find( jQuery object )**

version added: [1.6](#)

**jQuery object**

Type: [Object](#)

A jQuery object to match elements against.

**.find( element )**

version added: [1.6](#)

**element**

Type: [Element](#)

An element to match elements against.

Given a jQuery object that represents a set of DOM elements, the `.find()` method allows us to search through the descendants of these elements in the DOM tree and construct a new jQuery object from the matching elements. The `.find()` and `.children()` methods are similar, except that the latter only travels a single level down the DOM tree.

The first signature for the `.find()` method accepts a selector expression of the same type that we can pass to the `$( )` function. The elements will be filtered by testing whether they match this selector.

Consider a page with a basic nested list on it:

```

1 <ul class="level-1">
2   <li class="item-i">I</li>
3   <li class="item-ii">II
4     <ul class="level-2">
5       <li class="item-a">A</li>
6       <li class="item-b">B
7         <ul class="level-3">
8           <li class="item-1">1</li>
9           <li class="item-2">2</li>
10          <li class="item-3">3</li>
11        </ul>
12      </li>
13      <li class="item-c">C</li>
14    </ul>
15  </li>
16  <li class="item-iii">III</li>
17 </ul>

```

If we begin at item II, we can find list items within it:

```
1 | $('li.item-ii').find('li').css('background-co:
```



The result of this call is a red background on items A, B, 1, 2, 3, and C. Even though item II matches the selector expression, it is not included in the results; only descendants are considered candidates for the match.

*Unlike in the rest of the tree traversal methods, the selector expression is required in a call to `.find()`. If we need to retrieve all of the descendant elements, we can pass in the universal selector `'*'` to*

*accomplish this.*

Selector context is implemented with the `.find()` method; therefore, `$('.li.item-ii').find('li')` is equivalent to `$('.li', 'li.item-ii')`.

As of **jQuery 1.6**, we can also filter the selection with a given jQuery collection or element. With the same nested list as above, if we start with:

```
1 | var $allListElements = $('li');
```

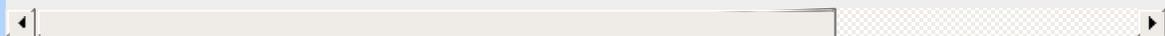
And then pass this jQuery object to find:

```
1 | $('.li.item-ii').find( $allListElements );
```

This will return a jQuery collection which contains only the list elements that are descendants of item II.

Similarly, an element may also be passed to find:

```
1 | var item1 = $('li.item-1')[0];  
2 | $('.li.item-ii').find( item1 ).css('background-
```



The result of this call would be a red background on item 1.

## Examples:

**Example:** Starts with all paragraphs and searches for descendant span elements, same as \$("p span")

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-
5 </head>
6 <body>
7   <p><span>Hello</span>, how are you?</p>
8 <p>Me? I'm <span>good</span>.</p>
9 <script>
10   $("p").find("span").css('color', 'red');
11 </script>
12
13 </body>
14 </html>
```

### Demo

**Example:** A selection using a jQuery collection of all span tags. Only spans within p tags are changed to red while others are left blue.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     span { color: blue; }
6   </style>
7   <script src="http://code.jquery.com/jquery-
8 </head>
```

```

9 <body>
10 <p><span>Hello</span>, how are you?</p>
11 <p>Me? I'm <span>good</span>.</p>
12 <div>Did you <span>eat</span> yet?</div>
13 <script>
14   var $spans = $('span');
15   $("p").find( $spans ).css('color','red');
16 </script>
17
18 </body>
19 </html>

```

## Demo

**Example:** Add spans around each word then add a hover and italicize words with the letter t.

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     p { font-size:20px; width:200px; cursor:default;
6         color:blue; font-weight:bold; margin:0; }
7     .hilite { background:yellow; }
8   </style>
9   <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
10 </head>
11 <body>
12   <p>
13     When the day is short
14     find that which matters to you
15     or stop believing
16   </p>
17 <script>
18   var newText = $("p").text().split(" ").join("<br>");
19   newText = "<span>" + newText + "</span>";

```

```
20
21   $("p").html( newText )
22   .find('span')
23   .hover(function() {
24       $(this).addClass("hilite");
25   },
26       function() { $(this).removeClass("hilite")
27   })
28   .end()
29   .find(":contains('t')")
30   .css({"font-style":"italic", "font-weight":
31
32 </script>
33
34 </body>
35 </html>
```

Demo

POWERED BY HERONOTE

A new version of this book is available!

---

# .finish()

Categories: [Effects](#) > [Custom](#)

`.finish( [queue ] )`

Returns: [jQuery](#)

**Description:** *Stop the currently-running animation, remove all queued animations, and complete all animations for the matched elements.*

`.finish( [queue ] )`

version added: [1.9](#)

**queue**

Type: [String](#)

The name of the queue in which to stop animations.

When `.finish()` is called on an element, the currently-running animation and all queued animations (if any) immediately stop and their CSS properties set to their target values. All queued animations are removed.

If the first argument is provided, only the animations in the queue represented by that string will be stopped.

The `.finish()` method is similar to `.stop(true, true)` in that it clears the queue and the current animation jumps to its end value. It differs, however, in that `.finish()` also causes the CSS property of all *queued* animations to jump to their end values, as well.

*Animations may be stopped globally by setting the property `$.fx.off` to `true`. When this is done, all animation methods will immediately set elements to their final state when called, rather than displaying an effect.*

## Example:

*Click the Go button once to start the animation and then click the other buttons to see how they affect the current and queued animations.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>.box {
5     position: absolute;
6     top: 10px;
7     left: 10px;
8     width: 15px;
9     height: 15px;
10    background: black;
11  }
12  #path {
13    height: 244px;
14    font-size: 70%;
15    border-left: 2px dashed red;
16    border-bottom: 2px dashed green;
17    border-right: 2px dashed blue;
18  }
19  button {
20    width: 12em;
21    display: block;
22    text-align: left;
23    margin: 0 auto;
24  }
25 </style>
26   <script src="http://code.jquery.com/jquery
27 </head>
28 <body>
29   <div class="box"></div>
```

```

30 <div id="path">
31   <button id="go">Go</button>
32   <br>
33   <button id="bstt" class="b">.stop(true, true)
34   <button id="bcf" class="b">.clearQueue().finish()
35   <br>
36   <button id="bstf" class="b">.stop(true, false)
37   <button id="bcs" class="b">.clearQueue().stop()
38   <br>
39   <button id="bsff" class="b">.stop(false, false)
40   <button id="bs" class="b">.stop()</button>
41   <br>
42   <button id="bsft" class="b">.stop(false, true)
43   <br>
44   <button id="bf" class="b">.finish()</button>
45 </div>
46
47 <script>
48 var horiz = $("#path").width() - 20,
49     vert = $("#path").height() - 20;
50
51 var btns = {
52   bstt: function () {
53     $("#div.box").stop(true, true);
54   },
55   bs: function () {
56     $("#div.box").stop();
57   },
58   bsft: function () {
59     $("#div.box").stop(false, true);
60   },
61   bf: function () {
62     $("#div.box").finish();
63   },
64   bcf: function () {
65     $("#div.box").clearQueue().finish();
66   },

```

```
67     bsff: function () {
68         $("div.box").stop(false, false);
69     },
70     bstf: function () {
71         $("div.box").stop(true, false);
72     },
73     bcs: function () {
74         $("div.box").clearQueue().stop();
75     }
76 };
77
78
79 $("button.b").on("click", function () {
80     btns[this.id]();
81 });
82
83 $("#go").on("click", function () {
84     $(".box")
85     .clearQueue()
86     .stop()
87     .css({
88     left: 10,
89     top: 10
90     })
91     .animate({
92     top: vert
93     }, 3000)
94     .animate({
95     left: horiz
96     }, 3000)
97     .animate({
98     top: 10
99     }, 3000);
100 });
101 </script>
102
103 </body>
```

104 | `</html>`

Demo

POWERED BY **HERONOTE**

A new version of this book is available!

---

# .first()

Categories: [Traversing](#) > [Filtering](#)

`.first()`

Returns: [jQuery](#)

**Description:** Reduce the set of matched elements to the first in the set.

`.first()`

version added: [1.4](#)

This method does not accept any arguments.

Given a jQuery object that represents a set of DOM elements, the `.first()` method constructs a new jQuery object from the first element in that set.

Consider a page with a simple list on it:

```
1 | <ul>
2 |   <li>list item 1</li>
3 |   <li>list item 2</li>
4 |   <li>list item 3</li>
5 |   <li>list item 4</li>
6 |   <li>list item 5</li>
7 | </ul>
```

We can apply this method to the set of list items:

```
1 | $('li').first().css('background-color', 'red')
```

The result of this call is a red background for the first item.

## Example:

*Highlight the first span in a paragraph.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>.highlight{background-color: yellow}.
5   <script src="http://code.jquery.com/jquery-
6 </head>
7 <body>
8   <p><span>Look:</span> <span>This is some te
9   <script>$("p span").first().addClass('highlig
10
11 </body>
12 </html>
```

Demo

POWERED BY **HERONOTE**

A new version of this book is available!

---

# :first-child Selector

Categories: [Selectors](#) > [Child Filter](#)

## first-child selector

**Description:** *Selects all elements that are the first child of their parent.*

**jQuery( ":first-child" )**

version added: [1.1.4](#)

While [:first](#) matches only a single element, the `:first-child` selector can match more than one: one for each parent. This is equivalent to `:nth-child(1)`.

## Example:

*Find the first span in each matched div to underline and add a hover state.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     span { color:#008; }
6     span.sogreen { color:green; font-weight: bo.
7   </style>
8   <script src="http://code.jquery.com/jquery-.
9 </head>
10 <body>
11   <div>
12     <span>John,</span>
13     <span>Karl,</span>
14     <span>Brandon</span>
15
16   </div>
17   <div>
18     <span>Glen,</span>
19     <span>Tane,</span>
20     <span>Ralph</span>
21
22   </div>
23 <script>
24   $("div span:first-child")
25     .css("text-decoration", "underline")
26     .hover(function () {
27       $(this).addClass("sogreen");
28     }, function () {
29       $(this).removeClass("sogreen");
30     });
```

```
31 </script>
32
33 </body>
34 </html>
```

Demo

POWERED BY **HERONOTE**

A new version of this book is available!

---

# :first-of-type Selector

Categories: [Selectors](#) > [Child Filter](#)

## first-of-type selector

**Description:** *Selects all elements that are the first among siblings of the same element name.*

**jQuery( ":first-of-type" )**

version added: [1.9](#)

The `:first-of-type` selector matches elements that have no other element with both the same parent and the same element name coming before it in the document tree.

## Example:

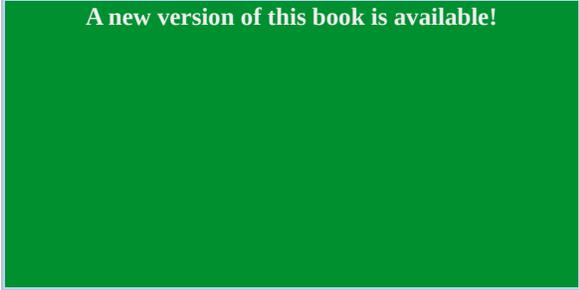
*Find the first span in each matched div and add a class to it.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5   span.fot { color: red; font-size: 120%; font-
6   </style>
7   <script src="http://code.jquery.com/jquery-
8 </head>
9 <body>
10  <div>
11    <span>Corey, </span>
12    <span>Yehuda, </span>
13    <span>Adam, </span>
14    <span>Todd</span>
15  </div>
16  <div>
17    <b>Nobody, </b>
18    <span>Jörn, </span>
19    <span>Scott, </span>
20    <span>Timo</span>
21  </div>
22  <script>
23  $("span:first-of-type").addClass("fot");
24  </script>
25
26 </body>
27 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

# :first Selector

Categories: [Selectors](#) > [Basic Filter](#) | [Selectors](#) > [jQuery Extensions](#)

## first selector

**Description:** *Selects the first matched element.*

**jQuery( ":first" )**

version added: [1.0](#)

The `:first` pseudo-class is equivalent to `:eq(0)`. It could also be written as `:lt(1)`. While this matches only a single element, [:first-child](#) can match more than one: One for each parent.

### Additional Notes:

Because `:first` is a jQuery extension and not part of the CSS specification, queries using `:first` cannot take advantage of the performance boost provided by the native DOM `querySelectorAll()` method. To achieve the best performance when using `:first` to select elements, first select the elements using a pure CSS selector, then use `.filter(":first")`.

Selected elements are in the order of their appearance in the document.

## Example:

*Finds the first table row.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5
6    td { color:blue; font-weight:bold; }
7  </style>
8  <script src="http://code.jquery.com/jquery-
9  </head>
10 <body>
11   <table>
12     <tr><td>Row 1</td></tr>
13     <tr><td>Row 2</td></tr>
14
15     <tr><td>Row 3</td></tr>
16   </table>
17   <script>$("tr:first").css("font-style", "ital.
18
19 </body>
20 </html>
```

Demo

**A new version of this book is available!**

---

# .focus()

Categories: [Events](#) > [Form Events](#) | [Forms](#)

**.focus( handler(eventObject) )**

Returns: [jQuery](#)

**Description:** *Bind an event handler to the "focus" JavaScript event, or trigger that event on an element.*

**.focus( handler(eventObject) )**

version added: [1.0](#)

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute each time the event is triggered.

**.focus( [eventData ],  
handler(eventObject) )**

version added: [1.4.3](#)

**eventData**

Type: [Object](#)

An object containing data that will be passed to the event handler.

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute each time the event is triggered.

**.focus()**

version added: [1.0](#)

This method does not accept any arguments.

This method is a shortcut for `.on('focus', handler)` in the first and second variations, and `.trigger('focus')` in the third.

The `focus` event is sent to an element when it gains focus. This event is implicitly applicable to a limited set of elements, such as form elements (`<input>`, `<select>`, etc.) and links (`<a href>`). In recent browser versions, the event can be extended to include all element types by explicitly setting the element's `tabindex` property. An element can gain focus via keyboard commands, such as the Tab key, or by mouse clicks on the element.

Elements with focus are usually highlighted in some way by the browser, for example with a dotted line surrounding the element.

The focus is used to determine which element is the first to receive keyboard-related events.

*Attempting to set focus to a hidden element causes an error in Internet Explorer. Take care to only use `.focus()` on elements that are visible. To run an element's focus event handlers without setting focus to the element, use `.triggerHandler("focus")` instead of `.focus()`.*

For example, consider the HTML:

```
1 | <form>
2 |   <input id="target" type="text" value="Field 1" />
3 |   <input type="text" value="Field 2" />
4 | </form>
5 | <div id="other">
6 |   Trigger the handler
7 | </div>
```

The event handler can be bound to the first input field:

```
1 | $('#target').focus(function() {
2 |   alert('Handler for .focus() called. ');
3 | });
```

Now clicking on the first field, or tabbing to it from another field, displays the alert:

Handler for `.focus()` called.

We can trigger the event when another element is clicked:

```
1 | $('#other').click(function() {  
2 |     $('#target').focus();  
3 | });
```

After this code executes, clicks on Trigger the handler will also alert the message.

The `focus` event does not bubble in Internet Explorer. Therefore, scripts that rely on event delegation with the `focus` event will not work consistently across browsers. As of version 1.4.2, however, jQuery works around this limitation by mapping `focus` to the `focusin` event in its event delegation methods, `.live()` and `.delegate()`.

## Examples:

**Example:** *Fire focus.*

```
1 | <!DOCTYPE html>
2 | <html>
3 | <head>
4 |   <style>span {display:none;}</style>
5 |   <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
6 | </head>
7 | <body>
8 |   <p><input type="text" /> <span>focus fire</span></p>
9 |   <p><input type="password" /> <span>focus fire</span></p>
10 | <script>
11 |   $("input").focus(function () {
12 |     $(this).next("span").css('display', 'inline');
13 |   });
14 | </script>
15 |
16 | </body>
17 | </html>
```

Demo

**Example:** *To stop people from writing in text input boxes, try:*

```
1 | $("input[type=text]").focus(function(){
2 |   $(this).blur();
3 | });
```

**Example:** *To focus on a login input box with id 'login' on page startup, try:*

```
1 | $(document).ready(function(){
2 |     $("#login").focus();
3 | });
```

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# :focus Selector

Categories: [Selectors](#) > [Basic Filter](#) | [Selectors](#) > [Form](#)

## focus selector

**Description:** *Selects element if it is currently focused.*

**jQuery( ":focus" )**

version added: [1.6](#)

As with other pseudo-class selectors (those that begin with a ":"), it is recommended to precede `:focus` with a tag name or some other selector; otherwise, the universal selector ("\*") is implied. In other words, the bare `$(':focus')` is equivalent to `$('*:focus')`. If you are looking for the currently focused element, `$( document.activeElement )` will retrieve it without having to search the whole DOM tree.

## Example:

*Adds the focused class to whatever element has focus*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     .focused {
6       background: #abcdef;
7     }
8   </style>
9   <script src="http://code.jquery.com/jquery-
10 </head>
11 <body>
12
13 <div id="content">
14   <input tabIndex="1">
15   <input tabIndex="2">
16   <select tabIndex="3">
17     <option>select menu</option>
18   </select>
19   <div tabIndex="4">
20     a div
21   </div>
22 </div>
23
24 <script>
25 $( "#content" ).delegate( "*", "focus blur",
26   var elem = $( this );
27   setTimeout(function() {
28     elem.toggleClass( "focused", elem.is(
29     }, 0);
30 });
```

```
31 </script>
32
33 </body>
34 </html>
```

Demo

POWERED BY **HERONOTE**

A new version of this book is available!

---

# .focusin()

Categories: [Events](#) > [Keyboard Events](#) | [Events](#) > [Mouse Events](#)

`.focusin( handler(eventObject) )`

Returns: [jQuery](#)

**Description:** *Bind an event handler to the "focusin" event.*

**.focusin( handler(eventObject) )**

version added: [1.4](#)

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute each time the event is triggered.

**.focusin( [eventData ],  
handler(eventObject) )**

version added: [1.4.3](#)

**eventData**

Type: [Object](#)

An object containing data that will be passed to the event handler.

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute each time the event is triggered.

This method is a shortcut for `.on('focusin', handler)`.

The `focusin` event is sent to an element when it, or any element inside of it, gains focus. This is distinct from the `focus` event in that it supports detecting the focus event on parent elements (in other words, it supports event bubbling).

This event will likely be used together with the `focusout` event.

## Example:

*Watch for a focus to occur within the paragraphs on the page.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>span {display:none;}</style>
5   <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
6 </head>
7 <body>
8   <p><input type="text" /> <span>focusin fire
9 <p><input type="password" /> <span>focusin fi
10 <script>
11 $( "p" ).focusin( function() {
12   $( this ).find( "span" ).css( 'display', 'inline'
13 });
14 </script>
15
16 </body>
17 </html>
```

Demo

POWERED BY **HERONOTE**

A new version of this book is available!

---

# .focusout()

Categories: [Events](#) > [Keyboard Events](#) | [Events](#) > [Mouse Events](#)

`.focusout( handler(eventObject) )`

Returns: [jQuery](#)

**Description:** *Bind an event handler to the "focusout" JavaScript event.*

`.focusout( handler(eventObject) )`

version added: [1.4](#)

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute each time the event is triggered.

`.focusout( [eventData ],  
handler(eventObject) )`

version added: [1.4.3](#)

**eventData**

Type: [Object](#)

An object containing data that will be passed to the event handler.

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute each time the event is triggered.

This method is a shortcut for `.on('focusout', handler)`.

The `focusout` event is sent to an element when it, or any element inside of it, loses focus. This is distinct from the [blur](#) event in that it supports detecting the loss of focus on descendant elements (in other words, it supports event bubbling).

This event will likely be used together with the [focusin](#) event.

## Example:

Watch for a loss of focus to occur inside paragraphs and note the difference between the `focusout` count and the `blur` count.

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5    .inputs { float: left; margin-right: 1em; }
6    .inputs p { margin-top: 0; }
7  </style>
8    <script src="http://code.jquery.com/jquery-
9  </head>
10 <body>
11
12 <div class="inputs">
13   <p>
14     <input type="text" /><br />
15     <input type="text" />
16   </p>
17   <p>
18     <input type="password" />
19   </p>
20 </div>
21 <div id="fo">focusout fire</div>
22 <div id="b">blur fire</div>
23
24 <script>
25 var fo = 0, b = 0;
26 $("p").focusout(function() {
27   fo++;
28   $("#fo")
29     .text("focusout fired: " + fo + "x");
```

```
30  }).blur(function() {  
31      b++;  
32      $("#b")  
33          .text("blur fired: " + b + "x");  
34  
35  });  
36  </script>  
37  
38  </body>  
39  </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# .get()

Categories: [Miscellaneous](#) > [DOM Element Methods](#)

`.get( [index ] )`

Returns: [Element, Array](#)

**Description:** Retrieve the DOM elements matched by the jQuery object.

`.get( [index ] )`

version added: [1.0](#)

**index**

Type: [Number](#)

A zero-based integer indicating which element to retrieve.

The `.get()` method grants us access to the DOM nodes underlying each jQuery object. Suppose we had a simple unordered list on the page:

```
1 | <ul>
2 |   <li id="foo">foo</li>
3 |   <li id="bar">bar</li>
4 | </ul>
```

Without a parameter, `.get()` returns all of the elements:

```
1 | alert($('li').get());
```

All of the matched DOM nodes are returned by this call, contained in a standard array:

```
[<li id="foo">, <li id="bar">]
```

With an index specified, `.get()` will retrieve a single element:

```
1 | ($('li').get(0));
```

Since the index is zero-based, the first list item is returned:

```
<li id="foo">
```

Each jQuery object also masquerades as an array, so we can use the array dereferencing operator to get at the list item instead:

```
1 | alert($('li')[0]);
```

However, this syntax lacks some of the additional capabilities of `.get()`, such as specifying a negative index:

```
1 | alert($('li').get(-1));
```

A negative index is counted from the end of the matched set, so this example will return the last item in the list:

```
<li id="bar">
```

---

## Examples:

**Example:** *Selects all divs in the document and returns the DOM Elements as an Array then uses the built-in reverse-method to reverse that array.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5      span { color:red; }
6    </style>
7    <script src="http://code.jquery.com/jquery-
8  </head>
9  <body>
10   Reversed - <span></span>
11
12   <div>One</div>
13   <div>Two</div>
14   <div>Three</div>
15  <script>
16  function disp(divs) {
17    var a = [];
18    for (var i = 0; i < divs.length; i++) {
19      a.push(divs[i].innerHTML);
20    }
21    $("span").text(a.join(" "));
22  }
23  disp( $("div").get().reverse() );
24  </script>
25
26  </body>
27  </html>
```

Demo

**Example:** Gives the tag name of the element clicked on.

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5    span { color:red; }
6    div { background:yellow; }
7  </style>
8    <script src="http://code.jquery.com/jquery-
9  </head>
10 <body>
11   <span>&nbsp;</span>
12   <p>In this paragraph is an <span>important<
13
14   <div><input type="text" /></div>
15 <script>
16 $("*", document.body).click(function (e) {
17   e.stopPropagation();
18   var domEl = $(this).get(0);
19   $("span:first").text("Clicked on - " + domE.
20 });
21 </script>
22
23 </body>
24 </html>
```

Demo

A new version of this book is available!

---

# :gt() Selector

Categories: [Selectors](#) > [Basic Filter](#) | [Selectors](#) > [jQuery Extensions](#)

## gt selector

**Description:** *Select all elements at an index greater than `index` within the matched set.*

**jQuery( ":gt(index)" )**

version added: [1.0](#)

**index:** Zero-based index.

### index-related selectors

The index-related selector expressions (including this "greater than" selector) filter the set of elements that have matched the expressions that precede them. They narrow the set down based on the order of the elements within this matched set. For example, if elements are first selected with a class selector (`.myclass`) and four elements are returned, these elements are given indices 0 through 3 for the purposes of these selectors.

Note that since JavaScript arrays use *0-based indexing*, these selectors reflect that fact. This is why `$('.myclass:gt(1)')` selects elements after the second element in the document with the class `myclass`, rather than after the first. In contrast, `:nth-child(n)` uses *1-based indexing* to conform to the CSS specification.

### Additional Notes:

Because `:gt()` is a jQuery extension and not part of the CSS specification, queries using `:gt()` cannot take advantage of the performance boost provided by the native DOM `querySelectorAll()` method. For better performance in modern browsers, use `$("your-pure-css-selector").slice(index)` instead.

## Example:

*Finds TD #5 and higher. Reminder: the indexing starts at 0.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <script src="http://code.jquery.com/jquery-...
5  </head>
6  <body>
7    <table border="1">
8
9      <tr><td>TD #0</td><td>TD #1</td><td>TD #2</td>
10     <tr><td>TD #3</td><td>TD #4</td><td>TD #5</td>
11
12     <tr><td>TD #6</td><td>TD #7</td><td>TD #8</td>
13   </table>
14   <script>$("td:gt(4)").css("text-decoration",
15
16   </body>
17   </html>
```

Demo

POWERED BY HERONOTE

A new version of this book is available!

---

# .has()

Categories: [Traversing](#) > [Filtering](#)

`.has( selector )`

Returns: [jQuery](#)

**Description:** Reduce the set of matched elements to those that have a descendant that matches the selector or DOM element.

`.has( selector )`

version added: [1.4](#)

**selector**

Type: [String](#)

A string containing a selector expression to match elements against.

`.has( contained )`

version added: [1.4](#)

**contained**

Type: [Element](#)

A DOM element to match elements against.

Given a jQuery object that represents a set of DOM elements, the `.has()` method constructs a new jQuery object from a subset of the matching elements. The supplied selector is tested against the descendants of the matching elements; the element will be included in the result if any of its descendant elements matches the selector.

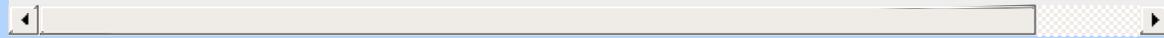
Consider a page with a nested list as follows:

```
1 | <ul>
2 |   <li>list item 1</li>
3 |   <li>list item 2
4 |     <ul>
5 |       <li>list item 2-a</li>
6 |       <li>list item 2-b</li>
7 |     </ul>
8 |   </li>
9 |   <li>list item 3</li>
10|   <li>list item 4</li>
```

```
11 | </ul>
```

We can apply this method to the set of list items as follows:

```
1 | $('li').has('ul').css('background-color', 'red')
```



The result of this call is a red background for item 2, as it is the only `<li>` that has a `<ul>` among its descendants.

## Example:

*Check if an element is inside another.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5      .full { border: 1px solid red; }
6    </style>
7    <script src="http://code.jquery.com/jquery-1.10.2.js"></script>
8  </head>
9  <body>
10
11 <ul><li>Does the UL contain an LI?</li></ul>
12
13 <script>
14   $("ul").append("<li>" + ($("ul").has("li")).length + " LI's");
15   $("ul").has("li").addClass("full");
16 </script>
17
18 </body>
19 </html>
```

Demo

**A new version of this book is available!**

---

# Has Attribute Selector [name]

Categories: [Selectors](#) > [Attribute](#)

## attributeHas selector

**Description:** *Selects elements that have the specified attribute, with any value.*

**jQuery( "[attribute]" )**

version added: [1.0](#)

**attribute:** An attribute name.

## Example:

*Bind a single click that adds the div id to its text.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-1.8.2.js"></script>
5 </head>
6 <body>
7   <div>no id</div>
8   <div id="hey">with id</div>
9
10  <div id="there">has an id</div>
11  <div>nope</div>
12  <script>
13    $('div[id]').one('click', function(){
14      var idString = $(this).text() + ' = ' + $(this).attr('id');
15      $(this).text(idString);
16    });
17  </script>
18
19 </body>
20 </html>
```

Demo

**A new version of this book is available!**

---

# :has() Selector

Categories: [Selectors](#) > [Content Filter](#) | [Selectors](#) > [jQuery Extensions](#)

## has selector

**Description:** *Selects elements which contain at least one element that matches the specified selector.*

**jQuery( ":has(selector)" )**

version added: [1.1.4](#)

**selector:** Any selector.

The expression `$('#div:has(p)')` matches a `<div>` if a `<p>` exists anywhere among its descendants, not just as a direct child.

### Additional Notes:

Because `:has()` is a jQuery extension and not part of the CSS specification, queries using `:has()` cannot take advantage of the performance boost provided by the native DOM `querySelectorAll()` method. For better performance in modern browsers, use `$("#your-pure-css-selector").has(selector/DOMElement)` instead.

## Example:

*Adds the class "test" to all divs that have a paragraph inside of them.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     .test{ border: 3px inset red; }
6   </style>
7   <script src="http://code.jquery.com/jquery-
8 </head>
9 <body>
10  <div><p>Hello in a paragraph</p></div>
11
12  <div>Hello again! (with no paragraph)</div>
13  <script>$("div:has(p)").addClass("test");</sc
14
15 </body>
16 </html>
```

Demo

POWERED BY HERONOTE

A new version of this book is available!

---

# .hasClass()

Categories: [Attributes](#) | [Manipulation](#) > [Class Attribute](#) | [CSS](#)

`.hasClass( className )`

Returns: [Boolean](#)

**Description:** Determine whether any of the matched elements are assigned the given class.

**.hasClass( className )**

version added: [1.2](#)

**className**

Type: [String](#)

The class name to search for.

Elements may have more than one class assigned to them. In HTML, this is represented by separating the class names with a space:

```
1 | <div id="mydiv" class="foo bar"></div>
```

The `.hasClass()` method will return `true` if the class is assigned to an element, even if other classes also are. For example, given the HTML above, the following will return `true`:

```
1 | $('#mydiv').hasClass('foo')
```

As would:

```
1 | $('#mydiv').hasClass('bar')
```

While this would return `false`:

```
1 | $('#mydiv').hasClass('quux')
```



## Example:

*Looks for the paragraph that contains 'selected' as a class.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     p { margin: 8px; font-size:16px; }
6     .selected { color:red; }
7   </style>
8   <script src="http://code.jquery.com/jquery-1.10.2.min.js"></script>
9 </head>
10 <body>
11
12   <p>This paragraph is black and is the first paragraph.
13   <p class="selected">This paragraph is red and is the last paragraph.
14
15   <div id="result1">First paragraph has selected class.
16   <div id="result2">Second paragraph has selected class.
17   <div id="result3">At least one paragraph has selected class.
18 </script>
19 $("div#result1").append($("#p:first").hasClass("selected"));
20 $("div#result2").append($("#p:last").hasClass("selected"));
21 $("div#result3").append($("#p").hasClass("selected"));
22 </script>
23
24 </body>
25 </html>
```

Demo

POWERED BY HERONOTE

A new version of this book is available!

---

# :header Selector

Categories: [Selectors](#) > [Basic Filter](#) | [Selectors](#) > [jQuery Extensions](#)

## header selector

**Description:** *Selects all elements that are headers, like h1, h2, h3 and so on.*

**jQuery( ":header" )**

version added: [1.2](#)

### Additional Notes:

Because `:header` is a jQuery extension and not part of the CSS specification, queries using `:header` cannot take advantage of the performance boost provided by the native DOM `querySelectorAll()` method. To achieve the best performance when using `:header` to select elements, first select the elements using a pure CSS selector, then use `.filter(":header")`.

## Example:

*Adds a background and text color to all the headers on the page.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <style>
5      body { font-size: 10px; font-family: Arial;
6      h1, h2 { margin: 3px 0; }
7      </style>
8      <script src="http://code.jquery.com/jquery-
9  </head>
10 <body>
11     <h1>Header 1</h1>
12
13     <p>Contents 1</p>
14     <h2>Header 2</h2>
15     <p>Contents 2</p>
16     <script>$(":header").css({ background: '#CCC',
17
18 </body>
19 </html>
```

Demo

**A new version of this book is available!**

---

# .height()

Categories: [CSS](#) | [Dimensions](#) | [Manipulation](#) > [Style Properties](#)

Get the current computed height for the first element in the set of matched elements or set the height of every matched element.

## Contents:

[.height\(\)](#)

[.height\(\)](#)

[.height\( value \)](#)

[.height\( value \)](#)

[.height\( function\(index, height\) \)](#)

---

`.height()`

Returns: `Integer`

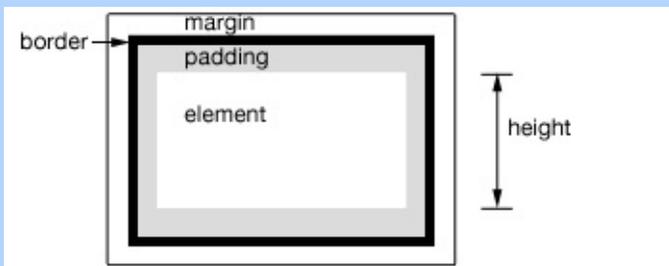
**Description:** *Get the current computed height for the first element in the set of matched elements.*

`.height()`

version added: `1.0`

This method does not accept any arguments.

The difference between `.css('height')` and `.height()` is that the latter returns a unit-less pixel value (for example, `400`) while the former returns a value with units intact (for example, `400px`). The `.height()` method is recommended when an element's height needs to be used in a mathematical calculation.



This method is also able to find the height of the window and document.

```
1 | $(window).height(); // returns height of browser window  
2 | $(document).height(); // returns height of HTML document
```

Note that `.height()` will always return the content height, regardless of the value of the CSS `box-sizing` property.

**Note:** Although `style` and `script` tags will report a value for `.width()` or `height()` when

*absolutely positioned and given `display:block`, it is strongly discouraged to call those methods on these tags. In addition to being a bad practice, the results may also prove unreliable.*



```
30     showHeight("document", $(document).height());
31     });
32     $("#getw").click(function () {
33         showHeight("window", $(window).height());
34     });
35
36 </script>
37
38 </body>
39 </html>
```

Demo

`.height( value )`

Returns: [jQuery](#)

**Description:** *Set the CSS height of every matched element.*

**`.height( value )`**

version added: [1.0](#)

**value**

Type: [String](#) or [Number](#)

An integer representing the number of pixels, or an integer with an optional unit of measure appended (as a string).

**`.height( function(index, height) )`**

version added: [1.4.1](#)

**function(index, height)**

Type: [Function\(\)](#)

A function returning the height to set. Receives the index position of the element in the set and the old height as arguments. Within the function, `this` refers to the current element in the set.

When calling `.height(value)`, the value can be either a string (number and unit) or a number. If only a number is provided for the value, jQuery assumes a pixel unit. If a string is provided, however, a valid CSS measurement must be provided for the height (such as `100px`, `50%`, or `auto`). Note that in modern browsers, the CSS height property does not include padding, border, or margin.

If no explicit unit was specified (like 'em' or '%') then "px" is concatenated to the value.

Note that `.height(value)` sets the content height of the box regardless of the value of the CSS `box-sizing` property.

## Example:

To set the height of each div on click to 30px plus a color change.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>div { width:50px; height:70px; float
5             background:rgb(255,140,0); cursor:poi
6   <script src="http://code.jquery.com/jquery-..
7 </head>
8 <body>
9   <div></div>
10  <div></div>
11
12  <div></div>
13  <div></div>
14  <div></div>
15 <script>$("div").one('click', function () {
16     $(this).height(30)
17     .css({cursor:"auto", backgroundCo
18     });</script>
19
20 </body>
21 </html>
```

Demo

**A new version of this book is available!**

---

# :hidden Selector

Categories: [Selectors](#) > [jQuery Extensions](#) | [Selectors](#) > [Visibility Filter](#)

## hidden selector

**Description:** *Selects all elements that are hidden.*

## jQuery( ":hidden" )

version added: [1.0](#)

Elements can be considered hidden for several reasons:

- They have a CSS `display` value of `none`.

- They are form elements with `type="hidden"`.

- Their width and height are explicitly set to 0.

- An ancestor element is hidden, so the element is not shown on the page.

Elements with `visibility: hidden` or `opacity: 0` are considered to be visible, since they still consume space in the layout. During animations that hide an element, the element is considered to be visible until the end of the animation. During animations to show an element, the element is considered to be visible at the start of the animation.

How `:hidden` is determined was changed in jQuery 1.3.2. An element is assumed to be hidden if it or any of its parents consumes no space in the document. CSS visibility isn't taken into account (therefore `$(elem).css('visibility','hidden').is(':hidden') == false`). The [release notes](#) outline the changes in more detail.

## Additional Notes:

Because `:hidden` is a jQuery extension and not part of the CSS specification, queries using `:hidden` cannot take advantage of the performance boost provided by the native DOM `querySelectorAll()` method. To achieve the best performance when using `:hidden` to select elements, first select the elements using a pure CSS selector, then use `.filter(":hidden")`.



## Example:

*Shows all hidden divs and counts hidden inputs.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     div { width:70px; height:40px; background:#f0f0f0; }
6     span { display:block; clear:left; color:red; }
7     .starthidden { display:none; }
8   </style>
9   <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
10 </head>
11 <body>
12   <span></span>
13   <div></div>
14   <div style="display:none;">Hider!</div>
15   <div></div>
16
17   <div class="starthidden">Hider!</div>
18   <div></div>
19   <form>
20     <input type="hidden" />
21
22     <input type="hidden" />
23     <input type="hidden" />
24   </form>
25   <span>
26
27   </span>
28 <script>
29 // in some browsers :hidden includes head, ti
30 var hiddenEls = $("body").find(":hidden").not("input");
31
```

```
32 | $("span:first").text("Found " + hiddenEls.length +
33 | $("div:hidden").show(3000);
34 | $("span:last").text("Found " + $("input:hidden"
35 | </script>
36 |
37 | </body>
38 | </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# .hide()

Categories: [Effects](#) > [Basics](#)

`.hide()`

Returns: [jQuery](#)

**Description:** *Hide the matched elements.*

**`.hide()`**

version added: [1.0](#)

This method does not accept any arguments.

**`.hide( [duration ] [, complete ] )`**

version added: [1.0](#)

**duration** (default: `400`)

Type: [Number](#) or [String](#)

A string or number determining how long the animation will run.

**complete**

Type: [Function\(\)](#)

A function to call once the animation is complete.

**`.hide( options )`**

version added: [1.0](#)

**options**

Type: [PlainObject](#)

A map of additional options to pass to the method.

**duration** (default: `400`)

Type: [Number](#) or [String](#)

A string or number determining how long the animation will run.

**easing** (default: `swing`)

Type: [String](#)

A string indicating which easing function to use for the transition.

**queue** (default: `true`)

Type: [Boolean](#)

A Boolean indicating whether to place the animation in the effects queue. If false, the animation will begin immediately.

**As of jQuery 1.7**, the queue option can also accept a string, in which case the animation is added to the queue represented by that string.

## specialEasing

Type: [PlainObject](#)

A map of one or more of the CSS properties defined by the properties argument and their corresponding easing functions. **(version added: 1.4)**

---

## step

Type: [Function](#)( [Number](#) now, [Tween](#) tween )

A function to be called for each animated property of each animated element. This function provides an opportunity to modify the Tween object to change the value of the property before it is set.

---

## progress

Type: [Function](#)( [Promise](#) animation, [Number](#) progress, [Number](#) remainingMs )

A function to be called after each step of the animation, only once per animated element regardless of the number of animated properties. **(version added: 1.8)**

---

## complete

Type: [Function](#)()

A function to call once the animation is complete.

---

## done

Type: [Function](#)( [Promise](#) animation, [Boolean](#) jumpedToEnd )

A function to be called when the animation completes (its Promise object is resolved). **(version added: 1.8)**

---

## fail

Type: [Function](#)( [Promise](#) animation, [Boolean](#) jumpedToEnd )

A function to be called when the animation fails to complete (its Promise object is rejected). **(version added: 1.8)**

---

## always

Type: [Function](#)( [Promise](#) animation, [Boolean](#) jumpedToEnd )

A function to be called when the animation completes or stops without completing (its Promise object is either resolved or rejected). **(version added: 1.8)**

## `.hide( [duration ] [, easing ] [, complete ] )`

version added: [1.4.3](#)

**duration** (default: `400`)

Type: [Number](#) or [String](#)

A string or number determining how long the animation will run.

**easing** (default: `swing`)

Type: [String](#)

A string indicating which easing function to use for the transition.

**complete**

Type: [Function\(\)](#)

A function to call once the animation is complete.

With no parameters, the `.hide()` method is the simplest way to hide an element:

```
1 | $(' .target' ).hide();
```

The matched elements will be hidden immediately, with no animation. This is roughly equivalent to calling `.css('display', 'none')`, except that the value of the `display` property is saved in jQuery's data cache so that `display` can later be restored to its initial value. If an element has a `display` value of `inline`, then is hidden and shown, it will once again be displayed `inline`.

When a duration, a plain object, or a "complete" function is provided, `.hide()` becomes an animation method. The `.hide()` method animates the width, height, and opacity of the matched elements simultaneously. When these properties reach 0, the `display` style property is set to `none` to ensure that the element no longer affects the layout of the page.

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings `'fast'` and `'slow'` can be supplied to indicate durations of `200` and `600` milliseconds, respectively.

Note that `.hide()` is fired immediately and will override the animation queue if no duration or a duration of 0 is specified.

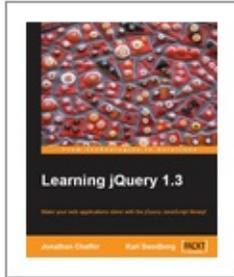
As of jQuery **1.4.3**, an optional string naming an easing function may be used. Easing functions specify the speed at which the animation progresses at different points within the animation. The only easing implementations in the jQuery library are the default, called `swing`, and one that progresses at a constant pace, called `linear`. More easing functions are available with the use of plug-ins, most notably the [jQuery UI suite](#).

If supplied, the callback is fired once the animation is complete. This can be useful for stringing different animations together in sequence. The callback is not sent any arguments, but `this` is set to the DOM element being animated. If multiple elements are animated, it is important to note that the callback is executed once per matched element, not once for the animation as a whole.

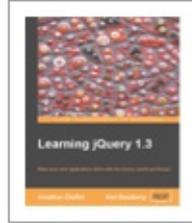
We can animate any element, such as a simple image:

```
1 <div id="clickme">
2   Click here
3 </div>
4 
5 With the element initially shown, we can hide it with the following code:
6 $('#clickme').click(function() {
7   $('#book').hide('slow', function() {
8     alert('Animation complete.');
```

Click here



Click here



Click here



Click here

Animation complete.

## Additional Notes:

All jQuery effects, including `.hide()`, can be turned off globally by setting `jquery.fx.off = true`, which effectively sets the duration to 0. For more information, see [jQuery.fx.off](#).

## Examples:

**Example:** *Hides all paragraphs then the link on click.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-
5 </head>
6 <body>
7   <p>Hello</p>
8   <a href="#">Click to hide me too</a>
9   <p>Here is another paragraph</p>
10 <script>
11   $("p").hide();
12   $("a").click(function ( event ) {
13     event.preventDefault();
14     $(this).hide();
15   });
16 </script>
17
18 </body>
19 </html>
```

Demo

**Example:** *Animates all shown paragraphs to hide slowly completing the animation within 600 milliseconds.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
```

```

5   p { background:#dad; font-weight:bold; }
6   </style>
7   <script src="http://code.jquery.com/jquery-latest.js"
8 </head>
9 <body>
10  <button>Hide 'em</button>
11
12  <p>Hiya</p>
13  <p>Such interesting text, eh?</p>
14  <script>
15      $("button").click(function () {
16          $("p").hide("slow");
17      });
18  </script>
19
20  </body>
21  </html>

```

## Demo

**Example:** *Animates all spans (words in this case) to hide fastly completing each animation within 200 milliseconds. Once each animation is done, it starts the next one.*

```

1   <!DOCTYPE html>
2   <html>
3   <head>
4       <style>
5           span { background:#def3ca; padding:3px; floo
6       </style>
7       <script src="http://code.jquery.com/jquery-
8   </head>
9   <body>
10      <button id="hidr">Hide</button>
11      <button id="showr">Show</button>

```

```

12 <div>
13
14 <span>Once</span> <span>upon</span> <span>a</
15 <span>time</span> <span>there</span> <span>were
16 <span>three</span> <span>programmers...</span>
17
18 </div>
19 <script>
20 $("#hidr").click(function () {
21     $("span:last-child").hide("fast", function ()
22         // use callee so don't have to name the function
23         $(this).prev().hide("fast", arguments.callee);
24     });
25 });
26 $("#showr").click(function () {
27     $("span").show(2000);
28 });
29 </script>
30
31 </body>
32 </html>

```

## Demo

**Example:** *Hides the divs when clicked over 2 seconds, then removes the div element when its hidden. Try clicking on more than one box at a time.*

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4 <style>
5 div { background:#ece023; width:30px;
6     height:40px; margin:2px; float:left;
7 </style>
8 <script src="http://code.jquery.com/jquery-

```

```
9 </head>
10 <body>
11   <div></div>
12 <script>
13   for (var i = 0; i < 5; i++) {
14     $("<div>").appendTo(document.body);
15   }
16   $("div").click(function () {
17     $(this).hide(2000, function () {
18       $(this).remove();
19     });
20   });
21 </script>
22
23 </body>
24 </html>
```

Demo

POWERED BY HERONOTE

A new version of this book is available!

---

# .hover()

Categories: [Events](#) > [Mouse Events](#)

Bind one or two handlers to the matched elements, to be executed when the mouse pointer enters and leaves the elements.

## Contents:

```
.hover\( handlerIn\(eventObject\), handlerOut\(eventObject\) \)  
  .hover( handlerIn(eventObject), handlerOut(eventObject) )  
  
.hover\( handlerInOut\(eventObject\) \)  
  .hover( handlerInOut(eventObject) )
```

`.hover( handlerIn(eventObject), handlerOut(eventObject) )`  
Returns: [jQuery](#)

**Description:** Bind two handlers to the matched elements, to be executed when the mouse pointer enters and leaves the elements.

`.hover( handlerIn(eventObject), handlerOut(eventObject) )`

version added: [1.0](#)

**handlerIn(eventObject)**

Type: [Function\(\)](#)

A function to execute when the mouse pointer enters the element.

**handlerOut(eventObject)**

Type: [Function\(\)](#)

A function to execute when the mouse pointer leaves the element.

The `.hover()` method binds handlers for both `mouseenter` and `mouseleave` events. You can use it to simply apply behavior to an element during the time the mouse is within the element.

Calling `$(selector).hover(handlerIn, handlerOut)` is shorthand for:

```
1 | $(selector).mouseenter(handlerIn).mouseleave(handlerOut)
```

See the discussions for [.mouseenter\(\)](#) and [.mouseleave\(\)](#) for more details.

## Examples:

**Example:** To add a special style to list items that are being hovered over, try:

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     ul { margin-left:20px; color:blue; }
6     li { cursor:default; }
7     span { color:red; }
8   </style>
9   <script src="http://code.jquery.com/jquery-
10 </head>
11 <body>
12   <ul>
13     <li>Milk</li>
14     <li>Bread</li>
15     <li class='fade'>Chips</li>
16
17     <li class='fade'>Socks</li>
18   </ul>
19 <script>
20 $( "li" ).hover(
21   function () {
22     $(this).append( $( "<span> ***</span>" ) );
23   },
24   function () {
25     $(this).find( "span:last" ).remove();
26   }
27 );
28
29 //li with fade class
30 $( "li.fade" ).hover( function() { $( this ).fadeOut
```

```
31 |
32 | </script>
33 |
34 | </body>
35 | </html>
```

## Demo

**Example:** *To add a special style to table cells that are being hovered over, try:*

```
1 | $("td").hover(
2 |   function () {
3 |     $(this).addClass("hover");
4 |   },
5 |   function () {
6 |     $(this).removeClass("hover");
7 |   }
8 | );
```

**Example:** *To unbind the above example use:*

```
1 | $("td").off('mouseenter mouseleave');
```

`.hover( handlerInOut(eventObject) )`

Returns: [jQuery](#)

**Description:** Bind a single handler to the matched elements, to be executed when the mouse pointer enters or leaves the elements.

`.hover( handlerInOut(eventObject) )`

version added: [1.4](#)

**handlerInOut(eventObject)**

Type: [Function\(\)](#)

A function to execute when the mouse pointer enters or leaves the element.

The `.hover()` method, when passed a single function, will execute that handler for both `mouseenter` and `mouseleave` events. This allows the user to use jQuery's various toggle methods within the handler or to respond differently within the handler depending on the `event.type`.

Calling `$(selector).hover(handlerInOut)` is shorthand for:

```
1 | $(selector).on("mouseenter mouseleave", handler)
```

See the discussions for `.mouseenter()` and `.mouseleave()` for more details.

## Example:

*Slide the next sibling LI up or down on hover, and toggle a class.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     ul { margin-left:20px; color:blue; }
6     li { cursor:default; }
7     li.active { background:black;color:white; }
8     span { color:red; }
9   </style>
10  <script src="http://code.jquery.com/jquery-
11 </head>
12 <body>
13   <ul>
14     <li>Milk</li>
15     <li>White</li>
16     <li>Carrots</li>
17     <li>Orange</li>
18     <li>Broccoli</li>
19     <li>Green</li>
20   </ul>
21 <script>
22 $( "li" )
23 .filter( ":odd" )
24 .hide()
25 .end()
26 .filter( ":even" )
27 .hover(
28   function () {
29     $(this).toggleClass( "active" )
30     .next().stop( true, true ).slideToggle();
```

```
31     }  
32   );  
33 </script>  
34  
35 </body>  
36 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# .html()

Categories: [Attributes](#) | [Manipulation](#) > [DOM Insertion, Inside](#)

Get the HTML contents of the first element in the set of matched elements or set the HTML contents of every matched element.

**Contents:**

[.html\(\)](#)

`.html()`

[.html\( htmlString \)](#)

`.html( htmlString )`

`.html( function(index, oldhtml) )`

---

`.html()`

Returns: [String](#)

**Description:** *Get the HTML contents of the first element in the set of matched elements.*

`.html()`

version added: [1.0](#)

This method does not accept any arguments.

This method is not available on XML documents.

In an HTML document, `.html()` can be used to get the contents of any element. If the selector expression matches more than one element, only the first match will have its HTML content returned. Consider this code:

```
1 | $('div.demo-container').html();
```

In order for the following `<div>`'s content to be retrieved, it would have to be the first one with `class="demo-container"` in the document:

```
1 | <div class="demo-container">
2 |   <div class="demo-box">Demonstration Box</div>
3 | </div>
```

The result would look like this:

```
1 | <div class="demo-box">Demonstration Box</div>
```

This method uses the browser's `innerHTML` property. Some browsers may not return HTML that exactly replicates the HTML source in an

original document. For example, Internet Explorer sometimes leaves off the quotes around attribute values if they contain only alphanumeric characters.

## Example:

*Click a paragraph to convert it from html to text.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5    p { margin:8px; font-size:20px; color:blue;
6      cursor:pointer; }
7    b { text-decoration:underline; }
8    button { cursor:pointer; }
9    </style>
10   <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
11 </head>
12 <body>
13   <p>
14     <b>Click</b> to change the <span id="tag">tag</span>
15   </p>
16   <p>
17     to a <span id="text">text</span> node.
18   </p>
19   <p>
20     This <button name="nada">button</button>
21   </p>
22 <script>
23 $( "p" ).click( function () {
24   var htmlStr = $( this ).html();
25   $( this ).text( htmlStr );
26 });
27 </script>
28
29 </body>
30 </html>
```



Demo

`.html( htmlString )`

Returns: [jQuery](#)

**Description:** *Set the HTML contents of each element in the set of matched elements.*

`.html( htmlString )`

version added: [1.0](#)

**htmlString**

Type: [htmlString](#)

A string of HTML to set as the content of each matched element.

`.html( function(index, oldhtml) )`

version added: [1.4](#)

**function(index, oldhtml)**

Type: [Function\(\)](#)

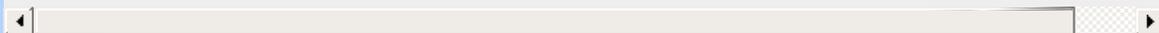
A function returning the HTML content to set. Receives the index position of the element in the set and the old HTML value as arguments. jQuery empties the element before calling the function; use the oldhtml argument to reference the previous content. Within the function, `this` refers to the current element in the set.

The `.html()` method is not available in XML documents.

When `.html()` is used to set an element's content, any content that was in that element is completely replaced by the new content. Additionally, jQuery removes other constructs such as data and event handlers from child elements before replacing those elements with the new content.

Consider the following HTML:

```
1 | <div class="demo-container">
2 |   <div class="demo-box">Demonstration Box</div>
3 | </div>
```



The content of `<div class="demo-container">` can be set like this:

```
1 | $( 'div.demo-container' )
2 |   .html( '<p>All new content. <em>You bet!</em>' );
```

That line of code will replace everything inside `<div class="demo-container">`:

```
1 | <div class="demo-container">
2 |   <p>All new content. <em>You bet!</em></p>
3 | </div>
```

As of jQuery 1.4, the `.html()` method allows the HTML content to be set by passing in a function.

```
1 | $( 'div.demo-container' ).html( function() {
2 |   var emph = '<em>' + $( 'p' ).length + ' paragraphs!</em>';
3 |   return '<p>All new content for ' + emph + '</p>';
4 | });
```

Given a document with six paragraphs, this example will set the HTML of `<div class="demo-container">` to `<p>All new content for <em>6 paragraphs!</em></p>`.

This method uses the browser's `innerHTML` property. Some browsers may not generate a DOM that exactly replicates the HTML source provided. For example, Internet Explorer prior to version 8 will convert all `href` properties on links to absolute URLs, and Internet Explorer prior to version 9 will not correctly handle HTML5 elements without the addition of a separate [compatibility layer](#).

**Note:** In Internet Explorer up to and including version 9, setting the text content of an HTML element may corrupt the text nodes of its children that are being removed from the document as a result of the

operation. If you are keeping references to these DOM elements and need them to be unchanged, use `.empty().html(string)` instead of `.html(string)` so that the elements are removed from the document before the new string is assigned to the element.

---

## Examples:

**Example:** *Add some html to each div.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     .red { color:red; }
6   </style>
7   <script src="http://code.jquery.com/jquery-1.10.2.min.js"></script>
8 </head>
9 <body>
10  <span>Hello</span>
11  <div></div>
12  <div></div>
13  <div></div>
14  <script>$("div").html("<span class='red'>Hello</span>");</script>
15
16 </body>
17 </html>
```

Demo

**Example:** *Add some html to each div then immediately do further manipulations to the inserted html.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     div { color:blue; font-size:18px; }
6   </style>
7   <script src="http://code.jquery.com/jquery-1.10.2.min.js"></script>
```

```
8 </head>
9 <body>
10 <div></div>
11 <div></div>
12 <div></div>
13 <script>
14     $("div").html("<b>Wow!</b> Such excitement...")
15     $("div b").append(document.createTextNode("!!!")
16                     .css("color", "red"));
17 </script>
18
19 </body>
20 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# ID Selector (“#id”)

Categories: [Selectors](#) > [Basic](#)

## id selector

**Description:** *Selects a single element with the given id attribute.*

**jQuery( "#id" )**

version added: [1.0](#)

**id:** An ID to search for, specified via the id attribute of an element.

For id selectors, jQuery uses the JavaScript function `document.getElementById()`, which is extremely efficient. When another selector is attached to the id selector, such as `h2#pageTitle`, jQuery performs an additional check before identifying the element as a match.

*As always, remember that as a developer, your time is typically the most valuable resource. Do not focus on optimization of selector speed unless it is clear that performance needs to be improved.*

Each `id` value must be used only once within a document. If more than one element has been assigned the same ID, queries that use that ID will only select the first matched element in the DOM. This behavior should not be relied on, however; a document with more than one element using the same ID is invalid.

If the id contains characters like periods or colons you have to [escape those characters with backslashes](#).

## Examples:

**Example:** Finds the element with the id "myDiv".

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5      div {
6        width: 90px;
7        height: 90px;
8        float:left;
9        padding: 5px;
10       margin: 5px;
11       background-color: #EEEEEE;
12     }
13   </style>
14   <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
15 </head>
16 <body>
17   <div id="notMe"><p>id="notMe"</p></div>
18
19   <div id="myDiv">id="myDiv"</div>
20 <script>$("#myDiv").css("border", "3px solid red");</script>
21
22 </body>
23 </html>
```

Demo

**Example:** Finds the element with the id "myDiv entry[1]".  
See how certain characters must be escaped with backslashes.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     div {
6       width: 300px;
7       float:left;
8       padding: 2px;
9       margin: 3px;
10      background-color: #EEEEEE;
11    }
12  </style>
13  <script src="http://code.jquery.com/jquery-latest.js">
14 </head>
15 <body>
16   <div id="myID.entry[0]">id="myID.entry[0]"</div>
17
18   <div id="myID.entry[1]">id="myID.entry[1]"</div>
19   <div id="myID.entry[2]">id="myID.entry[2]"</div>
20 <script>$("#myID\\.entry\\[1\\]").css("border",
21
22 </body>
23 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# :image Selector

Categories: [Selectors](#) > [Form](#) | [Selectors](#) > [jQuery Extensions](#)

## image selector

**Description:** *Selects all elements of type image.*

**jQuery( ":image" )**

version added: [1.0](#)

`:image` is equivalent to `[type="image"]`

### Additional Notes:

Because `:image` is a jQuery extension and not part of the CSS specification, queries using `:image` cannot take advantage of the performance boost provided by the native DOM `querySelectorAll()` method. For better performance in modern browsers, use `[type="image"]` instead.

## Example:

*Finds all image inputs.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5      textarea { height:45px; }
6    </style>
7    <script src="http://code.jquery.com/jquery-
8  </head>
9  <body>
10   <form>
11     <input type="button" value="Input Button" />
12     <input type="checkbox" />
13
14     <input type="file" />
15     <input type="hidden" />
16     <input type="image" />
17
18     <input type="password" />
19     <input type="radio" />
20     <input type="reset" />
21
22     <input type="submit" />
23     <input type="text" />
24     <select><option>Option</option></select>
25
26     <textarea></textarea>
27     <button>Button</button>
28   </form>
29   <div>
30 </div>
31 <script>
```

```
32 var input = $("input:image").css({background:"yellow"  
33 $("div").text("For this type jQuery found " + input.length  
34 .css("color", "red");  
35 $("form").submit(function () { return false; });  
36 </script>  
37  
38 </body>  
39 </html>
```

Demo

POWERED BYHERONOTE

A new version of this book is available!

---

# .index()

Categories: [Miscellaneous](#) > [DOM Element Methods](#)

`.index()`

Returns: [Number](#)

**Description:** Search for a given element from among the matched elements.

`.index()`

version added: [1.4](#)

This method does not accept any arguments.

`.index( selector )`

version added: [1.4](#)

**selector**

Type: [Selector](#)

A selector representing a jQuery collection in which to look for an element.

`.index( element )`

version added: [1.0](#)

**element**

Type: [Element](#) or [jQuery](#)

The DOM element or first element within the jQuery object to look for.

## Return Values

If no argument is passed to the `.index()` method, the return value is an integer indicating the position of the first element within the jQuery object relative to its sibling elements.

If `.index()` is called on a collection of elements and a DOM element or jQuery object is passed in, `.index()` returns an integer indicating the position of the passed element relative to the original collection.

If a selector string is passed as an argument, `.index()` returns an integer indicating the position of the first element within the jQuery object relative to the elements matched by the selector. If the element is not found, `.index()` will return -1.

## Detail

The complementary operation to `.get()`, which accepts an index and returns a DOM node, `.index()` can take a DOM node and returns an index. Suppose we have a simple unordered list on the page:

```
1 | <ul>
2 |   <li id="foo">foo</li>
3 |   <li id="bar">bar</li>
4 |   <li id="baz">baz</li>
5 | </ul>
```

If we retrieve one of the three list items (for example, through a DOM function or as the context to an event handler), `.index()` can search for this list item within the set of matched elements:

```
1 | var listItem = document.getElementById('bar');
2 | alert('Index: ' + $('li').index(listItem));
3 | We get back the zero-based position of the list item.
```

Index: 1

Similarly, if we retrieve a jQuery object consisting of one of the three list items, `.index()` will search for that list item:

```
1 | var listItem = $('#bar');
2 | alert('Index: ' + $('li').index(listItem));
```

We get back the zero-based position of the list item:

Index: 1

Note that if the jQuery collection used as the `.index()` method's argument contains more than one element, the first element within the matched set of elements will be used.

```
1 | var listItems = $('li:gt(0)');  
2 | alert('Index: ' + $('li').index(listItems));
```

We get back the zero-based position of the first list item within the matched set:

Index: 1

If we use a string as the `.index()` method's argument, it is interpreted as a jQuery selector string. The first element among the object's matched elements which also matches this selector is located.

```
1 | var listItem = $('#bar');  
2 | alert('Index: ' + listItem.index('li'));
```

We get back the zero-based position of the list item:

Index: 1

If we omit the argument, `.index()` will return the position of the first element within the set of matched elements in relation to its siblings:

```
1 | alert('Index: ' + $('#bar').index());
```

Again, we get back the zero-based position of the list item:

Index: 1

## Examples:

**Example:** *On click, returns the index (based zero) of that div in the page.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5  div { background:yellow; margin:5px; }
6  span { color:red; }
7  </style>
8    <script src="http://code.jquery.com/jquery-
9  </head>
10 <body>
11   <span>Click a div!</span>
12 <div>First div</div>
13 <div>Second div</div>
14 <div>Third div</div>
15 <script>
16 $("div").click(function () {
17   // this is the dom element clicked
18   var index = $("div").index(this);
19   $("span").text("That was div index #" + index);
20 });
21 </script>
22
23 </body>
24 </html>
```

Demo

**Example:** *Returns the index for the element with ID bar.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>div { font-weight: bold; color: #090;
5   <script src="http://code.jquery.com/jquery-latest.js"
6 </head>
7 <body>
8   <ul>
9     <li id="foo">foo</li>
10    <li id="bar">bar</li>
11    <li id="baz">baz</li>
12  </ul>
13  <div></div>
14  <script>var listItem = $('#bar');
15    $('#div').html( 'Index: ' + $('#li').index(listItem)
16
17 </body>
18 </html>
```

## Demo

**Example** Returns the index for the first item in the jQuery collection.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>div { font-weight: bold; color: #090
5   <script src="http://code.jquery.com/jquery-
6 </head>
7 <body>
8   <ul>
9     <li id="foo">foo</li>
10    <li id="bar">bar</li>
11    <li id="baz">baz</li>
12  </ul>
```

```
13 <div></div>
14 <script>var listItems = $('li:gt(0)');
15 $('div').html( 'Index: ' + $('li').index(listItems) );
16 </script>
17
18 </body>
19 </html>
```

### Demo

**Example** Returns the index for the element with ID bar in relation to all <li> elements.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>div { font-weight: bold; color: #090
5   <script src="http://code.jquery.com/jquery-
6 </head>
7 <body>
8   <ul>
9     <li id="foo">foo</li>
10    <li id="bar">bar</li>
11    <li id="baz">baz</li>
12  </ul>
13  <div></div>
14  <script>$('div').html('Index: ' + $('#bar')..
15
16 </body>
17 </html>
```

### Demo

**Example** Returns the index for the element with ID bar in relation to its siblings.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>div { font-weight: bold; color: #090;
5   <script src="http://code.jquery.com/jquery-latest.js"
6 </head>
7 <body>
8   <ul>
9     <li id="foo">foo</li>
10    <li id="bar">bar</li>
11    <li id="baz">baz</li>
12  </ul>
13  <div></div>
14  <script>var barIndex = $('#bar').index();
15  $('#div').html( 'Index: ' + barIndex );</script>
16
17 </body>
18 </html>
```

Demo

**Example** Returns -1, as there is no element with ID foobar.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>div { font-weight: bold; color: #090
5   <script src="http://code.jquery.com/jquery-
6 </head>
7 <body>
8   <ul>
9     <li id="foo">foo</li>
10    <li id="bar">bar</li>
11    <li id="baz">baz</li>
12  </ul>
```

```
13 <div></div>
14 <script>var foobar = $("li").index( $('#foobar'
15 $('div').html('Index: ' + foobar);</script>
16
17 </body>
18 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# .innerHeight()

Categories: [CSS](#) | [Dimensions](#) | [Manipulation](#) > [Style Properties](#)

`.innerHeight()`

Returns: *Integer*

**Description:** *Get the current computed height for the first element in the set of matched elements, including padding but not border.*

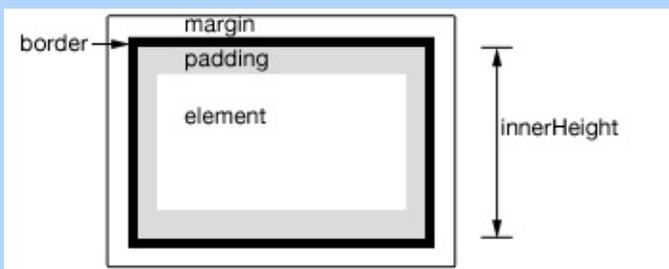
`.innerHeight()`

version added: [1.2.6](#)

This method does not accept any arguments.

This method returns the height of the element, including top and bottom padding, in pixels.

This method is not applicable to `window` and `document` objects; for these, use `.height()` instead.



## Example:

*Get the innerHeight of a paragraph.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>p { margin:10px;padding:5px;border:2px solid #ccc;}
5    <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
6  </head>
7  <body>
8    <p>Hello</p><p></p>
9    <script>var p = $("p:first");
10   $("p:last").text( "innerHeight:" + p.innerHeight() );
11
12  </body>
13  </html>
```

Demo

POWERED BY **HERONOTE**

A new version of this book is available!

---

# .innerWidth()

Categories: [CSS](#) | [Dimensions](#) | [Manipulation](#) > [Style Properties](#)

`.innerWidth()`

Returns: `Integer`

**Description:** *Get the current computed width for the first element in the set of matched elements, including padding but not border.*

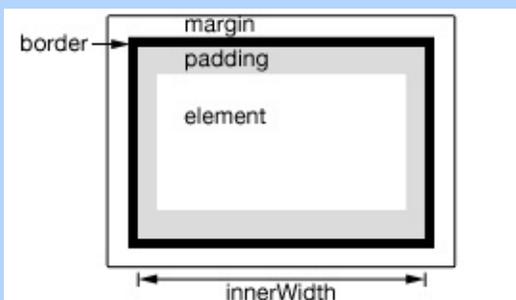
`.innerWidth()`

version added: `1.2.6`

This method does not accept any arguments.

This method returns the width of the element, including left and right padding, in pixels.

This method is not applicable to `window` and `document` objects; for these, use `.width()` instead.



## Example:

*Get the innerWidth of a paragraph.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <style>p { margin:10px;padding:5px;border:2px solid #ccc;}
5      <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
6  </head>
7  <body>
8      <p>Hello</p><p></p>
9      <script>var p = $("p:first");
10     $("p:last").text( "innerWidth:" + p.innerWidth() );
11
12 </body>
13 </html>
```

Demo

POWERED BY **HERONOTE**

A new version of this book is available!

---

# :input Selector

Categories: [Selectors](#) > [Form](#) | [Selectors](#) > [jQuery Extensions](#)

## input selector

**Description:** *Selects all input, textarea, select and button elements.*

**jQuery( ":input" )**

version added: [1.0](#)

The `:input` selector basically selects all form controls.

### Additional Notes:

Because `:input` is a jQuery extension and not part of the CSS specification, queries using `:input` cannot take advantage of the performance boost provided by the native DOM `querySelectorAll()` method. To achieve the best performance when using `:input` to select elements, first select the elements using a pure CSS selector, then use `.filter(":input")`.

## Example:

*Finds all input elements.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5      textarea { height:25px; }
6    </style>
7    <script src="http://code.jquery.com/jquery-
8  </head>
9  <body>
10   <form>
11     <input type="button" value="Input Button"
12     <input type="checkbox" />
13
14     <input type="file" />
15     <input type="hidden" />
16     <input type="image" />
17
18     <input type="password" />
19     <input type="radio" />
20     <input type="reset" />
21
22     <input type="submit" />
23     <input type="text" />
24     <select><option>Option</option></select>
25
26     <textarea></textarea>
27     <button>Button</button>
28   </form>
29   <div id="messages">
30   </div>
31 </script>
```

```
32 var allInputs = $(":input");
33 var formChildren = $("form > *");
34 $("#messages").text("Found " + allInputs.length +
35                       formChildren.length +
36
37 // so it won't submit
38 $("form").submit(function () { return false; });
39
40 </script>
41
42 </body>
43 </html>
```

Demo

POWERED BY HERONOTE

A new version of this book is available!

---

# .insertAfter()

Categories: [Manipulation](#) > [DOM Insertion, Outside](#)

`.insertAfter( target )`

Returns: `jQuery`

**Description:** *Insert every element in the set of matched elements after the target.*

`.insertAfter( target )`

version added: `1.0`

**target**

Type: `Selector` or `htmlString` or `Element` or `jQuery`

A selector, element, HTML string, or jQuery object; the matched set of elements will be inserted after the element(s) specified by this parameter.

The `.after()` and `.insertAfter()` methods perform the same task. The major difference is in the syntax-specifically, in the placement of the content and target. With `.after()`, the selector expression preceding the method is the container after which the content is inserted. With `.insertAfter()`, on the other hand, the content precedes the method, either as a selector expression or as markup created on the fly, and it is inserted after the target container.

Consider the following HTML:

```
1 | <div class="container">
2 |   <h2>Greetings</h2>
3 |   <div class="inner">Hello</div>
4 |   <div class="inner">Goodbye</div>
5 | </div>
```

We can create content and insert it after several elements at once:

```
1 | $('<p>Test</p>').insertAfter('.inner');
```

Each inner `<div>` element gets this new content:

```
1 | <div class="container">
2 |   <h2>Greetings</h2>
3 |   <div class="inner">Hello</div>
4 |   <p>Test</p>
5 |   <div class="inner">Goodbye</div>
6 |   <p>Test</p>
7 | </div>
```

We can also select an element on the page and insert it after another:

```
1 | $('h2').insertAfter($('.container'));
```

If an element selected this way is inserted into a single location elsewhere in the DOM, it will be moved after the target (not cloned):

```
1 | <div class="container">
2 |   <div class="inner">Hello</div>
3 |   <div class="inner">Goodbye</div>
4 | </div>
5 | <h2>Greetings</h2>
```

If there is more than one target element, however, cloned copies of the inserted element will be created for each target after the first, and that new set (the original element plus clones) is returned.

## Example:

*Insert all paragraphs after an element with id of "foo".  
Same as \$("#foo").after("p")*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>#foo { background:yellow; }</style>
5   <script src="http://code.jquery.com/jquery-
6 </head>
7 <body>
8   <p> is what I said... </p><div id="foo">FOO
9 <script>
10  $("#p").insertAfter("#foo");
11 </script>
12
13 </body>
14 </html>
```

Demo

POWERED BY HERONOTE

A new version of this book is available!

---

# .insertBefore()

Categories: [Manipulation](#) > [DOM Insertion, Outside](#)

`.insertBefore( target )`

Returns: [jQuery](#)

**Description:** *Insert every element in the set of matched elements before the target.*

`.insertBefore( target )`

version added: [1.0](#)

**target**

Type: [Selector](#) or [htmlString](#) or [Element](#) or [jQuery](#)

A selector, element, HTML string, or jQuery object; the matched set of elements will be inserted before the element(s) specified by this parameter.

The `.before()` and `.insertBefore()` methods perform the same task. The major difference is in the syntax-specifically, in the placement of the content and target. With `.before()`, the selector expression preceding the method is the container before which the content is inserted. With `.insertBefore()`, on the other hand, the content precedes the method, either as a selector expression or as markup created on the fly, and it is inserted before the target container.

Consider the following HTML:

```
1 | <div class="container">
2 |   <h2>Greetings</h2>
3 |   <div class="inner">Hello</div>
4 |   <div class="inner">Goodbye</div>
5 | </div>
```

We can create content and insert it before several elements at once:

```
1 | $( '<p>Test</p>' ).insertBefore( '.inner' );
```

Each inner `<div>` element gets this new content:

```
1 | <div class="container">
2 |   <h2>Greetings</h2>
3 |   <p>Test</p>
4 |   <div class="inner">Hello</div>
5 |   <p>Test</p>
6 |   <div class="inner">Goodbye</div>
7 | </div>
```

We can also select an element on the page and insert it before another:

```
1 | $('h2').insertBefore($('.container'));
```

If an element selected this way is inserted into a single location elsewhere in the DOM, it will be moved before the target (not cloned):

```
1 | <h2>Greetings</h2>
2 | <div class="container">
3 |   <div class="inner">Hello</div>
4 |   <div class="inner">Goodbye</div>
5 | </div>
```

If there is more than one target element, however, cloned copies of the inserted element will be created for each target after the first, and that new set (the original element plus clones) is returned.

## Example:

*Insert all paragraphs before an element with id of "foo". Same as \$("#foo").before("p")*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>#foo { background:yellow; }</style>
5   <script src="http://code.jquery.com/jquery-
6 </head>
7 <body>
8   <div id="foo">FOO!</div><p>I would like to
9 <script>
10 $("p").insertBefore("#foo");
11 </script>
12
13 </body>
14 </html>
```

Demo

POWERED BY HERONOTE

A new version of this book is available!

---

# .is()

Categories: [Traversing](#) > [Filtering](#)

`.is( selector )`

Returns: [Boolean](#)

**Description:** *Check the current matched set of elements against a selector, element, or jQuery object and return `true` if at least one of these elements matches the given arguments.*

`.is( selector )`

version added: [1.0](#)

**selector**

Type: [Selector](#)

A string containing a selector expression to match elements against.

`.is( function(index) )`

version added: [1.6](#)

**function(index)**

Type: [Function\(\)](#)

A function used as a test for the set of elements. It accepts one argument, `index`, which is the element's index in the jQuery collection. Within the function, `this` refers to the current DOM element.

`.is( jQuery object )`

version added: [1.6](#)

**jQuery object**

Type: [Object](#)

An existing jQuery object to match the current set of elements against.

`.is( element )`

version added: [1.6](#)

**element**

Type: [Element](#)

An element to match the current set of elements against.

Unlike other filtering methods, `.is()` does not create a new jQuery object. Instead, it allows you to test the contents of a jQuery object without modification. This is often useful inside callbacks, such as event handlers.

Suppose you have a list, with two of its items containing a child element:

```
1 | <ul>
2 |   <li>list <strong>item 1</strong></li>
3 |   <li><span>list item 2</span></li>
4 |   <li>list item 3</li>
5 | </ul>
```

You can attach a click handler to the `<ul>` element, and then limit the code to be triggered only when a list item itself, not one of its children, is clicked:

```
1 | $("ul").click(function(event) {
2 |   var $target = $(event.target);
3 |   if ( $target.is("li") ) {
4 |     $target.css("background-color", "red");
5 |   }
6 | });
```

Now, when the user clicks on the word "list" in the first item or anywhere in the third item, the clicked list item will be given a red background. However, when the user clicks on item 1 in the first item or anywhere in the second item, nothing will occur, because in those cases the target of the event would be `<strong>` or `<span>`, respectively.

Prior to jQuery 1.7, in selector strings with positional selectors such as `:first`, `:gt()`, or `:even`, the positional filtering is done against the jQuery object passed to `.is()`, *not* against the containing document. So for the HTML shown above, an expression such as `$("#li:first").is("li:last")` returns `true`, but `$("#li:first-child").is("li:last-child")` returns `false`. In addition, a bug in Sizzle prevented many positional selectors from working properly. These two factors made positional selectors almost unusable in filters.

Starting with jQuery 1.7, selector strings with positional selectors apply the selector against the document, and then determine whether the first element of the current jQuery set matches any of the resulting elements. So for the HTML shown above, an expression such as `$("#li:first").is("li:last")` returns `false`. Note that since positional selectors are jQuery additions and not W3C standard, we recommend using the W3C selectors whenever feasible.

## Using a Function

The second form of this method evaluates expressions related to elements based on a function rather than a selector. For each element, if the function returns `true`, `.is()` returns `true` as well. For example, given a somewhat more involved HTML snippet:

```
1 <ul>
2   <li><strong>list</strong> item 1 - one stron
3   <li><strong>list</strong> item <strong>2</st
4     two <span>strong tags</span></li>
5   <li>list item 3</li>
6   <li>list item 4</li>
7   <li>list item 5</li>
8 </ul>
```

You can attach a click handler to every `<li>` that evaluates the number of `<strong>` elements within the clicked `<li>` at that time like so:

```
1 $( "li" ).click( function() {
2   var $li = $( this ),
3     isWithTwo = $li.is( function() {
4       return $( 'strong', this ).length === 2;
5     } );
6   if ( isWithTwo ) {
7     $li.css( "background-color", "green" );
```

```
8     } else {  
9         $li.css("background-color", "red");  
10    }  
11 });
```

## Examples:

**Example:** Shows a few ways `is()` can be used inside an event handler.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     div { width:60px; height:60px; margin:5px;
6           border:4px outset; background:green; text-align:center;
7           font-weight:bolder; cursor:pointer; }
8     .blue { background:blue; }
9     .red { background:red; }
10    span { color:white; font-size:16px; }
11    p { color:red; font-weight:bolder; background-color:blue;
12        margin:3px; clear:left; display:none; }
13  </style>
14  <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
15 </head>
16 <body>
17   <div></div>
18   <div class="blue"></div>
19   <div></div>
20   <div class="red"></div>
21
22   <div><br/><span>Peter</span></div>
23   <div class="blue"></div>
24   <p>&nbsp;</p>
25   <script>
26     $("div").one('click', function () {
27       if ($(this).is(":first-child")) {
28         $("p").text("It's the first div.");
29       } else if ($(this).is(".blue,.red")) {
30         $("p").text("It's a blue or red div.");
```

```

31     } else if ($(this).is(":contains('Peter')")) {
32         $("p").text("It's Peter!");
33     } else {
34         $("p").html("It's nothing <em>special</em>.");
35     }
36     $("p").hide().slideDown("slow");
37     $(this).css({"border-style": "inset", cursor:
38 });
39 </script>
40
41 </body>
42 </html>

```

## Demo

**Example** Returns true, because the parent of the input is a form element.

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4     <style>div { color:red; }</style>
5     <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
6 </head>
7 <body>
8     <form><input type="checkbox" /></form>
9 </div></div>
10 <script>
11     var isFormParent = $("input[type='checkbox']").parent().is("form");
12     $("div").text("isFormParent = " + isFormParent);
13 </script>
14
15 </body>
16 </html>

```

## Demo

**Example:** Returns false, because the parent of the input is a p element.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>div { color:red; }</style>
5   <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
6 </head>
7 <body>
8   <form><p><input type="checkbox" /></p></form>
9   <div></div>
10 <script>
11   var isFormParent = $("input[type='checkbox']")
12     .parent().text("isFormParent = " + isFormParent);
13 </script>
14
15 </body>
16 </html>
```

## Demo

**Example:** Checks against an existing collection of alternating list elements. Blue alternating list elements slide up while others turn red.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>li { cursor:pointer; }</style>
5   <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
6 </head>
7 <body>
8
9 <ul id="browsers">
```

```

10     <li>Chrome</li>
11     <li>Safari</li>
12     <li>Firefox</li>
13     <li>Opera</li>
14 </ul>
15 <script>
16     var $alt = $("#browsers li:nth-child(2n)").css(
17     $('li').click(function() {
18         var $li = $(this);
19         if ( $li.is( $alt ) ) {
20             $li.slideUp();
21         } else {
22             $li.css( "background", "red" );
23         }
24     });
25 </script>
26
27 </body>
28 </html>

```

## Demo

**Example:** An alternate way to achieve the above example using an element rather than a jQuery object Checks against an existing collection of alternating list elements Blue alternating list elements slide up while others turn red.

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4     <style>li { cursor:pointer; }</style>
5     <script src="http://code.jquery.com/jquery-
6 </head>
7 <body>
8

```

```
 9 <ul id="browsers">
10   <li>Chrome</li>
11   <li>Safari</li>
12   <li>Firefox</li>
13   <li>Opera</li>
14 </ul>
15 <script>
16   var $alt = $("#browsers li:nth-child(2n)").css(
17     $('li').click(function() {
18       if ( $alt.is( this ) ) {
19         $(this).slideUp();
20       } else {
21         $(this).css("background", "red");
22       }
23     });
24 </script>
25
26 </body>
27 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# jQuery()

Categories: [Core](#)

Return a collection of matched elements either found in the DOM based on passed argument(s) or created by passing an HTML string.

## Contents:

[jQuery\( selector \[, context \] \)](#)

jQuery( selector [, context ] )

jQuery( element )

jQuery( elementArray )

jQuery( object )

jQuery( jQuery object )

jQuery()

[jQuery\( html \[, ownerDocument \] \)](#)

jQuery( html [, ownerDocument ] )

jQuery( html, attributes )

[jQuery\( callback \)](#)

jQuery( callback )

`jQuery( selector [, context ] )`

Returns: [jQuery](#)

**Description:** *Accepts a string containing a CSS selector which is then used to match a set of elements.*

**jQuery( selector [, context ] )**

version added: [1.0](#)

**selector**

Type: [selector](#)

A string containing a selector expression

**context**

Type: [Element](#) or [jQuery](#)

A DOM Element, Document, or jQuery to use as context

**jQuery( element )**

version added: [1.0](#)

**element**

Type: [Element](#)

A DOM element to wrap in a jQuery object.

**jQuery( elementArray )**

version added: [1.0](#)

**elementArray**

Type: [Array](#)

An array containing a set of DOM elements to wrap in a jQuery object.

**jQuery( object )**

version added: [1.0](#)

**object**

Type: [PlainObject](#)

A plain object to wrap in a jQuery object.

**jQuery( jQuery object )**

version added: [1.0](#)

**jQuery object**

Type: [PlainObject](#)

An existing jQuery object to clone.

This method does not accept any arguments.

In the first formulation listed above, `jQuery()` — which can also be written as `$( )` — searches through the DOM for any elements that match the provided selector and creates a new jQuery object that references these elements:

```
1 | $( "div.foo" );
```

If no elements match the provided selector, the new jQuery object is "empty"; that is, it contains no elements and has `.length` property of 0.

### Selector Context

By default, selectors perform their searches within the DOM starting at the document root. However, an alternate context can be given for the search by using the optional second parameter to the `$( )` function. For example, to do a search within an event handler, the search can be restricted like so:

```
1 | $( "div.foo" ).click(function() {  
2 |     $( "span", this ).addClass( "bar" );  
3 | });
```

When the search for the `span` selector is restricted to the context of `this`, only spans within the clicked element will get the additional class.

Internally, selector context is implemented with the `.find()` method, so `$( "span", this )` is equivalent to `$( this ).find( "span" )`.

### Using DOM elements

The second and third formulations of this function create a jQuery object using one or more DOM elements that were already selected

in some other way.

**Note:** These formulations are meant to consume only DOM elements; feeding mixed data to the `elementArray` form is particularly discouraged.

A common use of this facility is to call jQuery methods on an element that has been passed to a callback function through the keyword `this`:

```
1 | $( "div.foo" ).click(function() {  
2 |     $(this).slideUp();  
3 | });
```

This example causes elements to be hidden with a sliding animation when clicked. Because the handler receives the clicked item in the `this` keyword as a bare DOM element, the element must be passed to the `$( )` function before applying jQuery methods to it.

XML data returned from an Ajax call can be passed to the `$( )` function so individual elements of the XML structure can be retrieved using `.find()` and other DOM traversal methods.

```
1 | $.post( "url.xml", function(data) {  
2 |     var $child = $(data).find("child");  
3 | })
```

## Cloning jQuery Objects

When a jQuery object is passed to the `$( )` function, a clone of the object is created. This new jQuery object references the same DOM elements as the initial one.

## Returning an Empty Set

As of jQuery 1.4, calling the `jquery()` method with *no arguments* returns an empty jQuery set (with a `.length` property of 0). In previous versions of jQuery, this would return a set containing the

document node.

## Working With Plain Objects

At present, the only operations supported on plain JavaScript objects wrapped in jQuery are: `.data()`, `.prop()`, `.on()`, `.off()`, `.trigger()` and `.triggerHandler()`. The use of `.data()` (or any method requiring `.data()`) on a plain object will result in a new property on the object called `jQuery{randomNumber}` (eg. `jQuery123456789`).

```
1 // define a plain object
2 var foo = {foo: "bar", hello: "world"};
3
4 // Pass it to the jQuery function
5 var $foo = $( foo );
6
7 // test accessing property values
8 var test1 = $foo.prop( "foo" ); // bar
9
10 // test setting property values
11 $foo.prop( "foo", "foobar" );
12 var test2 = $foo.prop( "foo" ); // foobar
13
14 // test using .data() as summarized above
15 $foo.data( "keyName", "someValue" );
16 console.log( $foo ); // will now contain a j
17
18 // test binding an event name and triggering
19 $foo.on( "eventName", function () {
20     console.log("eventName was called");
21 });
22
23 $foo.trigger( "eventName" ); // logs "eventNa
```

Should `.trigger( "eventName" )` be used, it will search for an

"eventName" property on the object and attempt to execute it after any attached jQuery handlers are executed. It does not check whether the property is a function or not. To avoid this behavior, `.triggerHandler( "eventName" )` should be used instead.

```
1 | $foo.triggerHandler( "eventName" ); // also lo
```

## Examples:

**Example:** Find all *n* elements that are children of a *div* element and apply a border to them.

```
1 | <!DOCTYPE html>
2 | <html>
3 | <head>
4 |   <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
5 | </head>
6 | <body>
7 |   <p>one</p> <div><p>two</p></div> <p>three</p>
8 | <script>
9 |   $( "div > p" ).css( "border", "1px solid gray" );
10 | </script>
11 |
12 | </body>
13 | </html>
```

### Demo

**Example:** Find all *inputs* of type *radio* within the first *form* in the document.

```
1 | $("input:radio", document.forms[0]);
```

**Example:** Find all *div* elements within an *XML* document from an *Ajax* response.

```
1 | $("div", xml.responseXML);
```

**Example:** Set the background color of the page to

***black.***

```
1 | $(document.body).css( "background", "black" );
```

**Example:** *Hide all the input elements within a form.*

```
1 | $(myForm.elements).hide()
```

`jQuery( html [, ownerDocument ] )`

Returns: [jQuery](#)

**Description:** *Creates DOM elements on the fly from the provided string of raw HTML.*

`jQuery( html [, ownerDocument ] )`

version added: [1.0](#)

**html**

Type: [htmlString](#)

A string of HTML to create on the fly. Note that this parses HTML, **not** XML.

**ownerDocument**

Type: [document](#)

A document in which the new elements will be created

`jQuery( html, attributes )`

version added: [1.4](#)

**html**

Type: [htmlString](#)

A string defining a single, standalone, HTML element (e.g. `<div/>` or `<div></div>`).

**attributes**

Type: [PlainObject](#)

An object of attributes, events, and methods to call on the newly-created element.

## Creating New Elements

If a string is passed as the parameter to `$()`, jQuery examines the string to see if it looks like HTML (i.e., it starts with `<tag ... >`). If not, the string is interpreted as a selector expression, as explained above. But if the string appears to be an HTML snippet, jQuery attempts to create new DOM elements as described by the HTML. Then a jQuery object is created and returned that refers to these elements. You can perform any of the usual jQuery methods on this object:

```
1 | $( "<p id='test'>My <em>new</em> text</p>" ).appendTo(
```

For explicit parsing of a string to HTML, use the [\\$.parseHTML\(\)](#) method.

If the HTML is more complex than a single tag without attributes, as it is in the above example, the actual creation of the elements is handled by the browser's `innerHTML` mechanism. In most cases, jQuery creates a new `<div>` element and sets the `innerHTML` property of the element to the HTML snippet that was passed in. When the parameter has a single tag (with optional closing tag or quick-closing) — `$( "<img />" )` or `$( "<img>" )`, `$( "<a></a>" )` or `$( "<a>" )` — jQuery creates the element using the native JavaScript `createElement()` function.

When passing in complex HTML, some browsers may not generate a DOM that exactly replicates the HTML source provided. As mentioned, jQuery uses the browser's `.innerHTML` property to parse the passed HTML and insert it into the current document. During this process, some browsers filter out certain elements such as `<html>`, `<title>`, or `<head>` elements. As a result, the elements inserted may not be representative of the original string passed.

Filtering isn't, however, limited to these tags. For example, Internet Explorer prior to version 8 will also convert all `href` properties on links to absolute URLs, and Internet Explorer prior to version 9 will not correctly handle HTML5 elements without the addition of a separate [compatibility layer](#).

To ensure cross-platform compatibility, the snippet must be well-formed. Tags that can contain other elements should be paired with a closing tag:

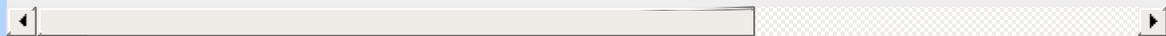
```
1 | $( "<a href='http://jquery.com'></a>" );
```

Tags that cannot contain elements may be quick-closed or not:

```
1 | $( "<img />" );
2 | $( "<input>" );
```

When passing HTML to `jquery()`, please also note that text nodes are not treated as DOM elements. With the exception of a few methods (such as `.content()`), they are generally otherwise ignored or removed. E.g:

```
1 | var el = $( "1<br/>2<br/>3" ); // returns [<br>
2 | el = $( "1<br/>2<br/>3 >" ); // returns [<br>
```



This behavior is expected.

As of jQuery 1.4, the second argument to `jquery()` can accept a plain object consisting of a superset of the properties that can be passed to the `.attr()` method.

**Important:** If the second argument is passed, the HTML string in the first argument must represent a simple element with no attributes.

**As of jQuery 1.4**, any [event type](#) can be passed in, and the following jQuery methods can be called: [val](#), [css](#), [html](#), [text](#), [data](#), [width](#), [height](#), or [offset](#).

**As of jQuery 1.8**, any jQuery instance method (a method of `jquery.fn`) can be used as a property of the object passed to the second parameter:

```
1 | $( "<div></div>", {
2 |   "class": "my-div",
3 |   on: {
4 |     touchstart: function( event ) {
5 |       // do something
6 |     }
7 |   }
8 | }).appendTo( "body" );
```

The name `"class"` must be quoted in the object since it is a JavaScript reserved word, and `"className"` cannot be used since it refers to the DOM property, not the attribute.

While the second argument is convenient, its flexibility can lead to unintended consequences (e.g. `$("<input>", {size: "4"})` calling the `.size()` method instead of setting the size attribute). The previous code block could thus be written instead as:

```
1 | $( "<div></div>" )
2 |   .addClass( "my-div" )
3 |   .on({
4 |     touchstart: function( event ) {
5 |       // do something
6 |     }
7 |   }).appendTo( "body" );
```

## Examples:

**Example:** Create a div element (and all of its contents) dynamically and append it to the body element. Internally, an element is created and its innerHTML property set to the given markup.

```
1 | $( "<div><p>Hello</p></div>" ).appendTo( "body"
```

**Example:** Create some DOM elements.

```
1 | $( "<div/>", {  
2 |   "class": "test",  
3 |   text: "Click me!",  
4 |   click: function() {  
5 |     $( this ).toggleClass( "test" );  
6 |   }  
7 | }).appendTo( "body" );
```

jQuery( callback )

Returns: jQuery

**Description:** *Binds a function to be executed when the DOM has finished loading.*

jQuery( callback )

version added: [1.0](#)

**callback**

Type: [Function\(\)](#)

The function to execute when the DOM is ready.

This function behaves just like `$(document).ready()`, in that it should be used to wrap other `$( )` operations on your page that depend on the DOM being ready. While this function is, technically, chainable, there really isn't much use for chaining against it.

## Examples:

**Example:** *Execute the function when the DOM is ready to be used.*

```
1 | $(function(){  
2 |   // Document is ready  
3 | });
```

**Example:** *Use both the shortcut for \$(document).ready() and the argument to write failsafe jQuery code using the \$ alias, without relying on the global alias.*

```
1 | jQuery(function($) {  
2 |   // Your code using failsafe $ alias here...  
3 | });
```

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# .jquery

Categories: [Internals](#) | [Properties](#) > [Properties of jQuery Object Instances](#)

`jquery`

Returns: `String`

**Description:** *A string containing the jQuery version number.*

`jquery`

version added: `1.0`

The `.jquery` property is assigned to the jQuery prototype, commonly referred to by its alias `$.fn`. It is a string containing the version number of `jquery`, such as "1.5.0" or "1.4.4".

## Examples:

**Example:** *Determine if an object is a jQuery object*

```
1 | var a = { what: "A regular JS object" },
2 |     b = $('body');
3 |
4 | if ( a.jquery ) { // falsy, since it's undefi
5 |     alert(' a is a jQuery object! ');
6 | }
7 |
8 | if ( b.jquery ) { // truthy, since it's a str.
9 |     alert(' b is a jQuery object! ');
10 | }
```

**Example:** *Get the current version of jQuery running on the page*

```
1 | alert( 'You are running jQuery version: ' + $.
```

POWERED BY HERONOTE

A new version of this book is available!

---

# jQuery.ajax()

Categories: [Ajax](#) > [Low-Level Interface](#)

**jQuery.ajax( url [, settings ] )**

Returns: [jqXHR](#)

**Description:** *Perform an asynchronous HTTP (Ajax) request.*

**jQuery.ajax( url [, settings ] )**

version added: [1.5](#)

**url**

Type: [String](#)

A string containing the URL to which the request is sent.

**settings**

Type: [PlainObject](#)

A set of key/value pairs that configure the Ajax request. All settings are optional. A default can be set for any option with [\\$.ajaxSetup\(\)](#). See [jQuery.ajax\( settings \)](#) below for a complete list of all settings.

**jQuery.ajax( [settings ] )**

version added: [1.0](#)

**settings**

Type: [PlainObject](#)

A set of key/value pairs that configure the Ajax request. All settings are optional. A default can be set for any option with [\\$.ajaxSetup\(\)](#).

**accepts** (default: `depends on DataType`)

Type: [PlainObject](#)

The content type sent in the request header that tells the server what kind of response it will accept in return. If the `accepts` setting needs modification, it is recommended to do so once in the `$.ajaxSetup()` method.

**async** (default: `true`)

Type: [Boolean](#)

By default, all requests are sent asynchronously (i.e. this is set to `true` by default). If you need synchronous requests, set this option to `false`. Cross-domain requests and `dataType: "jsonp"` requests do not support synchronous operation. Note that synchronous requests may temporarily lock the browser, disabling any actions while the request is

**active.** As of jQuery 1.8, the use of `async: false` with `jqXHR ($.Deferred)` is deprecated; you must use the **success/error/complete** callback options instead of the corresponding methods of the `jqXHR` object such as `jqXHR.done()` or the deprecated `jqXHR.success()`.

---

### **beforeSend**

Type: [Function](#)( `jqXHR jqXHR`, [PlainObject settings](#) )

A pre-request callback function that can be used to modify the `jqXHR` (in jQuery 1.4.x, `XMLHttpRequest`) object before it is sent. Use this to set custom headers, etc. The `jqXHR` and settings objects are passed as arguments. This is an [Ajax Event](#). Returning `false` in the `beforeSend` function will cancel the request. **As of jQuery 1.5**, the `beforeSend` option will be called regardless of the type of request.

---

**cache** (default: `true`, `false` for `dataType 'script'` and `'jsonp'`)

Type: [Boolean](#)

If set to `false`, it will force requested pages not to be cached by the browser. **Note:** Setting `cache` to `false` will only work correctly with HEAD and GET requests. It works by appending `"_{timestamp}"` to the GET parameters. The parameter is not needed for other types of requests, except in IE8 when a POST is made to a URL that has already been requested by a GET.

---

### **complete**

Type: [Function](#)( `jqXHR jqXHR`, [String textStatus](#) )

A function to be called when the request finishes (after `success` and `error` callbacks are executed). The function gets passed two arguments: The `jqXHR` (in jQuery 1.4.x, `XMLHttpRequest`) object and a string categorizing the status of the request (`"success"`, `"notmodified"`, `"error"`, `"timeout"`, `"abort"`, or `"parsererror"`). **As of jQuery 1.5**, the `complete` setting can accept an array of functions. Each function will be called in turn. This is an [Ajax Event](#).

---

### **contents**

Type: [PlainObject](#)

An object of string/regular-expression pairs that determine how jQuery will parse the response, given its content type. **(version added: 1.5)**

**contentType** (default: `'application/x-www-form-urlencoded; charset=UTF-8'`)

Type: [String](#)

When sending data to the server, use this content type. Default is "application/x-www-form-urlencoded; charset=UTF-8", which is fine for most cases. If you explicitly pass in a content-type to `$.ajax()`, then it'll always be sent to the server (even if no data is sent). If no charset is specified, data will be transmitted to the server using the server's default charset; you must decode this appropriately on the server side.

**context**

Type: [PlainObject](#)

This object will be made the context of all Ajax-related callbacks. By default, the context is an object that represents the ajax settings used in the call (`$.ajaxSettings` merged with the settings passed to `$.ajax`). For example, specifying a DOM element as the context will make that the context for the `complete` callback of a request, like so:

```
1 | $.ajax({
2 |   url: "test.html",
3 |   context: document.body
4 | }).done(function() {
5 |   $(this).addClass("done");
6 | });
```

**converters** (default: `{ "* text": window.String, "text html": true, "text json": jQuery.parseJSON, "text xml": jQuery.parseXML }`)

Type: [PlainObject](#)

An object containing dataType-to-dataType converters. Each converter's value is a function that returns the transformed value of the response. (**version added:** [1.5](#))

**crossDomain** (default: `false` for same-domain requests, `true` for cross-domain requests)

Type: [Boolean](#)

If you wish to force a crossDomain request (such as JSONP) on the same domain, set the value of crossDomain to `true`. This allows, for example, server-side redirection to another domain. **(version added: [1.5](#))**

---

### **data**

Type: [PlainObject](#) or [String](#)

Data to be sent to the server. It is converted to a query string, if not already a string. It's appended to the url for GET-requests. See `processData` option to prevent this automatic processing. Object must be Key/Value pairs. If value is an Array, jQuery serializes multiple values with same key based on the value of the `traditional` setting (described below).

---

### **dataFilter**

Type: [Function](#)( [PlainObject](#) data, [String](#) type ) => [Object](#)

A function to be used to handle the raw response data of XMLHttpRequest. This is a pre-filtering function to sanitize the response. You should return the sanitized data. The function accepts two arguments: The raw data returned from the server and the 'dataType' parameter.

---

**dataType** (default: `Intelligent Guess (xml, json, script, or html)`)

Type: [String](#)

The type of data that you're expecting back from the server. If none is specified, jQuery will try to infer it based on the MIME type of the response (an XML MIME type will yield XML, in 1.4 JSON will yield a JavaScript object, in 1.4 script will execute the script, and anything else will be returned as a string). The available types (and the result passed as the first argument to your success callback) are:

"xml": Returns a XML document that can be processed via jQuery.

---

"html": Returns HTML as plain text; included script tags are evaluated when inserted in the DOM.

---

"script": Evaluates the response as JavaScript and returns it as plain text. Disables caching by appending a query string parameter, "`_[TIMESTAMP]`", to the

URL unless the `cache` option is set to `true`. **Note:** This will turn POSTs into GETs for remote-domain requests.

"json": Evaluates the response as JSON and returns a JavaScript object. In jQuery 1.4 the JSON data is parsed in a strict manner; any malformed JSON is rejected and a parse error is thrown. (See [json.org](http://json.org) for more information on proper JSON formatting.)

"jsonp": Loads in a JSON block using [JSONP](#). Adds an extra "?callback=?" to the end of your URL to specify the callback. Disables caching by appending a query string parameter, "\_=[TIMESTAMP]", to the URL unless the `cache` option is set to `true`.

"text": A plain text string.

multiple, space-separated values: **As of jQuery 1.5**, jQuery can convert a `dataType` from what it received in the Content-Type header to what you require. For example, if you want a text response to be treated as XML, use "text xml" for the `dataType`. You can also make a JSONP request, have it received as text, and interpreted by jQuery as XML: "jsonp text xml." Similarly, a shorthand string such as "jsonp xml" will first attempt to convert from jsonp to xml, and, failing that, convert from jsonp to text, and then from text to xml.

## error

Type: [Function](#)( [jqXHR](#) jqXHR, [String](#) textStatus, [String](#) errorThrown )

A function to be called if the request fails. The function receives three arguments: The `jqXHR` (in jQuery 1.4.x, `XMLHttpRequest`) object, a string describing the type of error that occurred and an optional exception object, if one occurred. Possible values for the second argument (besides `null`) are "timeout", "error", "abort", and "parsererror". When an HTTP error occurs, `errorThrown` receives the textual portion of the HTTP status, such as "Not Found" or "Internal Server Error." **As of jQuery 1.5**, the `error` setting can accept an array of functions. Each function will be called in turn. **Note:** *This handler is not*

called for cross-domain script and JSONP requests. This is an [Ajax Event](#).

**global** (default: `true`)

Type: [Boolean](#)

Whether to trigger global Ajax event handlers for this request. The default is `true`. Set to `false` to prevent the global handlers like `ajaxStart` or `ajaxStop` from being triggered. This can be used to control various [Ajax Events](#).

**headers** (default: `{}`)

Type: [PlainObject](#)

An object of additional header key/value pairs to send along with the request. This setting is set before the `beforeSend` function is called; therefore, any values in the `headers` setting can be overwritten from within the `beforeSend` function. **(version added: [1.5](#))**

**ifModified** (default: `false`)

Type: [Boolean](#)

Allow the request to be successful only if the response has changed since the last request. This is done by checking the Last-Modified header. Default value is `false`, ignoring the header. In jQuery 1.4 this technique also checks the 'etag' specified by the server to catch unmodified data.

**isLocal** (default: `depends on current location protocol`)

Type: [Boolean](#)

Allow the current environment to be recognized as "local," (e.g. the filesystem), even if jQuery does not recognize it as such by default. The following protocols are currently recognized as local: `file`, `*-extension`, and `widget`. If the `isLocal` setting needs modification, it is recommended to do so once in the `$.ajaxSetup()` method. **(version added: [1.5.1](#))**

**jsonp**

Type: [String](#)

Override the callback function name in a jsonp request.

This value will be used instead of 'callback' in the 'callback=?' part of the query string in the url. So

`{jsonp: 'onJSONPLoad'}` would result in `'onJSONPLoad=?'`

passed to the server. **As of jQuery 1.5**, setting the `jsonp`

option to `false` prevents jQuery from adding the "?callback" string to the URL or attempting to use "=?" for transformation. In this case, you should also explicitly set the `jsonpCallback` setting. For example, `{ jsonp: false, jsonpCallback: "callbackName" }`

---

### **jsonpCallback**

Type: [String](#) or [Function\(\)](#)

Specify the callback function name for a JSONP request. This value will be used instead of the random name automatically generated by jQuery. It is preferable to let jQuery generate a unique name as it'll make it easier to manage the requests and provide callbacks and error handling. You may want to specify the callback when you want to enable better browser caching of GET requests. **As of jQuery 1.5**, you can also use a function for this setting, in which case the value of `jsonpCallback` is set to the return value of that function.

---

### **contentType**

Type: [String](#)

A mime type to override the [XHR](#) mime type. (**version added: 1.5.1**)

---

### **password**

Type: [String](#)

A password to be used in response to an HTTP access authentication request.

---

### **processData** (default: `true`)

Type: [Boolean](#)

By default, data passed in to the `data` option as an object (technically, anything other than a string) will be processed and transformed into a query string, fitting to the default content-type "application/x-www-form-urlencoded". If you want to send a `DOMDocument`, or other non-processed data, set this option to `false`.

---

### **scriptCharset**

Type: [String](#)

Only applies when the "script" transport is used (e.g., cross-domain requests with "jsonp" or "script" dataType and "GET" type). Sets the `charset` attribute on the script tag used in the request. Used when the character set on

the local page is not the same as the one on the remote script.

### **statusCode** (default: `{}`)

Type: [PlainObject](#)

An object of numeric HTTP codes and functions to be called when the response has the corresponding code. For example, the following will alert when the response status is a 404:

```
1 | $.ajax({
2 |     statusCode: {
3 |         404: function() {
4 |             alert("page not found");
5 |         }
6 |     }
7 | });
```

If the request is successful, the status code functions take the same parameters as the success callback; if it results in an error, they take the same parameters as the `error` callback.

(version added: [1.5](#))

### **success**

Type: [Function](#)( [PlainObject](#) data, [String](#) textStatus, [jqXHR](#) jqXHR )

A function to be called if the request succeeds. The function gets passed three arguments: The data returned from the server, formatted according to the `dataType` parameter; a string describing the status; and the `jqXHR` (in jQuery 1.4.x, XMLHttpRequest) object. **As of jQuery 1.5**, the success setting can accept an array of functions. Each function will be called in turn. This is an [Ajax Event](#).

### **timeout**

Type: [Number](#)

Set a timeout (in milliseconds) for the request. This will override any global timeout set with `$.ajaxSetup()`. The timeout period starts at the point the `$.ajax` call is made; if

several other requests are in progress and the browser has no connections available, it is possible for a request to time out before it can be sent. **In jQuery 1.4.x and below**, the XMLHttpRequest object will be in an invalid state if the request times out; accessing any object members may throw an exception. **In Firefox 3.0+ only**, script and JSONP requests cannot be cancelled by a timeout; the script will run even if it arrives after the timeout period.

---

### **traditional**

Type: [Boolean](#)

Set this to `true` if you wish to use the traditional style of [param serialization](#).

---

### **type** (default: `'GET'`)

Type: [String](#)

The type of request to make ("POST" or "GET"), default is "GET". **Note:** Other HTTP request methods, such as PUT and DELETE, can also be used here, but they are not supported by all browsers.

---

### **url** (default: `The current page`)

Type: [String](#)

A string containing the URL to which the request is sent.

---

### **username**

Type: [String](#)

A username to be used in response to an HTTP access authentication request.

---

### **xhr** (default: `ActiveXObject when available (IE), the XMLHttpRequest otherwise`)

Type: [Function\(\)](#)

Callback for creating the XMLHttpRequest object. Defaults to the ActiveXObject when available (IE), the XMLHttpRequest otherwise. Override to provide your own implementation for XMLHttpRequest or enhancements to the factory.

---

### **xhrFields**

Type: [PlainObject](#)

An object of fieldName-fieldValue pairs to set on the native `XHR` object. For example, you can use it to set `withCredentials` to `true` for cross-domain requests if

needed.

```
1 | $.ajax({
2 |     url: a_cross_domain_url,
3 |     xhrFields: {
4 |         withCredentials: true
5 |     }
6 | });
```

In **jQuery 1.5**, the `withCredentials` property was not propagated to the native `XHR` and thus CORS requests requiring it would ignore this flag. For this reason, we recommend using jQuery 1.5.1+ should you require the use of it.

**(version added: [1.5.1](#))**

The `$.ajax()` function underlies all Ajax requests sent by jQuery. It is often unnecessary to directly call this function, as several higher-level alternatives like `$.get()` and `.load()` are available and are easier to use. If less common options are required, though, `$.ajax()` can be used more flexibly.

At its simplest, the `$.ajax()` function can be called with no arguments:

```
1 | $.ajax();
```

**Note:** Default settings can be set globally by using the `$.ajaxSetup()` function.

This example, using no options, loads the contents of the current page, but does nothing with the result. To use the result, you can implement one of the callback functions.

## The jqXHR Object

The jQuery XMLHttpRequest (jqXHR) object returned by `$.ajax()`

as of jQuery 1.5 is a superset of the browser's native XMLHttpRequest object. For example, it contains `responseText` and `responseXML` properties, as well as a `getResponseHeader()` method. When the transport mechanism is something other than XMLHttpRequest (for example, a script tag for a JSONP request) the `jqXHR` object simulates native XHR functionality where possible.

As of jQuery 1.5.1, the `jqXHR` object also contains the `overrideMimeType()` method (it was available in jQuery 1.4.x, as well, but was temporarily removed in jQuery 1.5). The `.overrideMimeType()` method may be used in the `beforeSend()` callback function, for example, to modify the response content-type header:

```
1 | $.ajax({
2 |   url: "http://fiddle.jshell.net/favicon.png"
3 |   beforeSend: function ( xhr ) {
4 |     xhr.overrideMimeType("text/plain; charset=
5 |   }
6 | }).done(function ( data ) {
7 |   if( console && console.log ) {
8 |     console.log("Sample of data:", data.slice
9 |   }
10 | });
```

The `jqXHR` objects returned by `$.ajax()` as of jQuery 1.5 implement the Promise interface, giving them all the properties, methods, and behavior of a Promise (see [Deferred object](#) for more information). These methods take one or more function arguments that are called when the `$.ajax()` request terminates. This allows you to assign multiple callbacks on a single request, and even to assign callbacks after the request may have completed. (If the request is already complete, the callback is fired immediately.) Available Promise methods of the `jqXHR` object include:

**`jqXHR.done(function(data, textStatus, jqXHR) {});`**

An alternative construct to the success callback option, the `.done()` method replaces the deprecated `jqXHR.success()`

method. Refer to `deferred.done()` for implementation details.

**`jqXHR.fail(function(jqXHR, textStatus, errorThrown) {});`**  
An alternative construct to the error callback option, the `.fail()` method replaces the deprecated `.error()` method. Refer to `deferred.fail()` for implementation details.

**`jqXHR.always(function(data|jqXHR, textStatus, jqXHR|errorThrown) { });`**  
An alternative construct to the complete callback option, the `.always()` method replaces the deprecated `.complete()` method.

In response to a successful request, the function's arguments are the same as those of `.done()`: `data`, `textStatus`, and the `jqXHR` object. For failed requests the arguments are the same as those of `.fail()`: the `jqXHR` object, `textStatus`, and `errorThrown`. Refer to `deferred.always()` for implementation details.

**`jqXHR.then(function(data, textStatus, jqXHR) {}, function(jqXHR, textStatus, errorThrown) {});`**  
Incorporates the functionality of the `.done()` and `.fail()` methods, allowing (as of jQuery 1.8) the underlying Promise to be manipulated. Refer to `deferred.then()` for implementation details.

**Deprecation Notice:** *The `jqXHR.success()`, `jqXHR.error()`, and `jqXHR.complete()` callbacks are deprecated as of jQuery 1.8. To prepare your code for their eventual removal, use `jqXHR.done()`, `jqXHR.fail()`, and `jqXHR.always()` instead.*

```
1 // Assign handlers immediately after making request
2 // and remember the jqxhr object for this request
```

```
3 var jqxhr = $.ajax( "example.php" )
4   .done(function() { alert("success"); })
5   .fail(function() { alert("error"); })
6   .always(function() { alert("complete"); });
7
8 // perform other work here ...
9
10 // Set another completion function for the request above
11 jqxhr.always(function() { alert("second complete"); });
```

The `this` reference within all callbacks is the object in the `context` option passed to `$.ajax` in the settings; if `context` is not specified, `this` is a reference to the Ajax settings themselves.

For backward compatibility with `XMLHttpRequest`, a `jqXHR` object will expose the following properties and methods:

`readyState`

`status`

`statusText`

`responseXML` and/or `responseText` when the underlying request responded with xml and/or text, respectively

`setRequestHeader(name, value)` which departs from the standard by replacing the old value with the new one rather than concatenating the new value to the old one

`getAllResponseHeaders()`

`getResponseHeader()`

`statusCode()`

`abort()`

No `onreadystatechange` mechanism is provided, however, since `done`, `fail`, `always`, and `statusCode` cover all conceivable requirements.

## Callback Function Queues

The `beforeSend`, `error`, `dataFilter`, `success` and `complete` options all

accept callback functions that are invoked at the appropriate times.

As of jQuery 1.5, the `fail` and `done`, and, as of jQuery 1.6, `always` callback hooks are first-in, first-out managed queues, allowing for more than one callback for each hook. See [Deferred object methods](#), which are implemented internally for these `$.ajax()` callback hooks.

The callback hooks provided by `$.ajax()` are as follows:

- 1 `beforeSend` callback option is invoked; it receives the `jqXHR` object and the `settings` object as parameters.
- 2 `error` callback option is invoked, if the request fails. It receives the `jqXHR`, a string indicating the error type, and an exception object if applicable. Some built-in errors will provide a string as the exception object: "abort", "timeout", "No Transport".
- 3 `dataFilter` callback option is invoked immediately upon successful receipt of response data. It receives the returned data and the value of `dataType`, and must return the (possibly altered) data to pass on to `success`.
- 4 `success` callback option is invoked, if the request succeeds. It receives the returned data, a string containing the success code, and the `jqXHR` object.
- 5 **Promise callbacks** — `.done()`, `.fail()`, `.always()`, and `.then()` — are invoked, in the order they are registered.
- 6 `complete` callback option fires, when the request finishes, whether in failure or success. It receives the `jqXHR` object, as well as a string containing the success or error code.

## Data Types

The `$.ajax()` function relies on the server to provide information about the retrieved data. If the server reports the return data as XML, the result can be traversed using normal XML methods or jQuery's selectors. If another type is detected, such as HTML in the example above, the data is treated as text.

Different data handling can be achieved by using the `dataType` option. Besides plain `xml`, the `dataType` can be `html`, `json`, `jsonp`, `script`, or `text`.

The `text` and `xml` types return the data with no processing. The data is simply passed on to the success handler, either through the `responseText` or `responseXML` property of the `jqXHR` object, respectively.

**Note:** We must ensure that the MIME type reported by the web server matches our choice of `dataType`. In particular, XML must be declared by the server as `text/xml` or `application/xml` for consistent results.

If `html` is specified, any embedded JavaScript inside the retrieved data is executed before the HTML is returned as a string. Similarly, `script` will execute the JavaScript that is pulled back from the server, then return nothing.

The `json` type parses the fetched data file as a JavaScript object and returns the constructed object as the result data. To do so, it uses `jQuery.parseJSON()` when the browser supports it; otherwise it uses a `Function constructor`. Malformed JSON data will throw a parse error (see [json.org](http://json.org) for more information). JSON data is convenient for communicating structured data in a way that is concise and easy for JavaScript to parse. If the fetched data file exists on a remote server, specify the `jsonp` type instead.

The `jsonp` type appends a query string parameter of `callback=?` to the URL. The server should prepend the JSON data with the callback name to form a valid JSONP response. We can specify a parameter name other than `callback` with the `jsonp` option to `$.ajax()`.

**Note:** JSONP is an extension of the JSON format, requiring some server-side code to detect and handle the query string parameter. More information about it can be found in the [original post detailing its use](#).

When data is retrieved from remote servers (which is only possible using the `script` or `jsonp` data types), the `error` callbacks and global events will never be fired.

## Sending Data to the Server

By default, Ajax requests are sent using the GET HTTP method. If the POST method is required, the method can be specified by

setting a value for the `type` option. This option affects how the contents of the `data` option are sent to the server. POST data will always be transmitted to the server using UTF-8 charset, per the W3C XMLHttpRequest standard.

The `data` option can contain either a query string of the form `key1=value1&key2=value2`, or an object of the form `{key1: 'value1', key2: 'value2'}`. If the latter form is used, the data is converted into a query string using `jQuery.param()` before it is sent. This processing can be circumvented by setting `processData` to `false`. The processing might be undesirable if you wish to send an XML object to the server; in this case, change the `contentType` option from `application/x-www-form-urlencoded` to a more appropriate MIME type.

## Advanced Options

The `global` option prevents handlers registered using `.ajaxSend()`, `.ajaxError()`, and similar methods from firing when this request would trigger them. This can be useful to, for example, suppress a loading indicator that was implemented with `.ajaxSend()` if the requests are frequent and brief. With cross-domain script and JSONP requests, the `global` option is automatically set to `false`. See the descriptions of these methods below for more details. See the descriptions of these methods below for more details.

If the server performs HTTP authentication before providing a response, the user name and password pair can be sent via the `username` and `password` options.

Ajax requests are time-limited, so errors can be caught and handled to provide a better user experience. Request timeouts are usually either left at their default or set as a global default using `$.ajaxSetup()` rather than being overridden for specific requests with the `timeout` option.

By default, requests are always issued, but the browser may serve results out of its cache. To disallow use of the cached results, set `cache` to `false`. To cause the request to report failure if the asset has not been modified since the last request, set `ifModified` to `true`.

The `scriptCharset` allows the character set to be explicitly specified for requests that use a `<script>` tag (that is, a type of `script` or

`jsonp`). This is useful if the script and host page have differing character sets.

The first letter in Ajax stands for "asynchronous," meaning that the operation occurs in parallel and the order of completion is not guaranteed. The `async` option to `$.ajax()` defaults to `true`, indicating that code execution can continue after the request is made. Setting this option to `false` (and thus making the call no longer asynchronous) is strongly discouraged, as it can cause the browser to become unresponsive.

The `$.ajax()` function returns the `XMLHttpRequest` object that it creates. Normally jQuery handles the creation of this object internally, but a custom function for manufacturing one can be specified using the `xhr` option. The returned object can generally be discarded, but does provide a lower-level interface for observing and manipulating the request. In particular, calling `.abort()` on the object will halt the request before it completes.

**At present**, due to a bug in Firefox where `.getAllResponseHeaders()` returns the empty string although `.getResponseHeader('Content-Type')` returns a non-empty string, automatically decoding JSON CORS responses in Firefox with jQuery is not supported.

A workaround to this is possible by overriding `jQuery.ajaxSettings.xhr` as follows:

```
1  var _super = jQuery.ajaxSettings.xhr;
2  jQuery.ajaxSettings.xhr = function () {
3    var xhr = _super(),
4        getAllResponseHeaders = xhr.getAllResponseHeaders;
5
6    xhr.getAllResponseHeaders = function () {
7      if ( getAllResponseHeaders() ) {
8        return getAllResponseHeaders();
9      }
10     var allHeaders = "";
11
12     $( [ "Cache-Control", "Content-Language",
13         "Expires", "Last-Modified", "Pragma"
```

```

14     .each(function (i, header_name) {
15         if ( xhr.getResponseHeader( header_name ) ) {
16             allHeaders += header_name + ": " + xhr.getRespo
17         }
18         return allHeaders;
19     });
20 };
21
22 return xhr;
23 };

```

## Extending Ajax

As of jQuery 1.5, jQuery's Ajax implementation includes [prefilters](#), [transports](#), and converters that allow you to extend Ajax with a great deal of flexibility.

## Using Converters

`$.ajax()` converters support mapping data types to other data types. If, however, you want to map a custom data type to a known type (e.g `json`), you must add a correspondance between the response Content-Type and the actual data type using the `contents` option:

```

1  $.ajaxSetup({
2    contents: {
3      mycustomtype: /mycustomtype/
4    },
5    converters: {
6      "mycustomtype json": function ( result )
7        // do stuff
8        return newresult;
9    }
10  }
11  });

```

This extra object is necessary because the response Content-Types and data types never have a strict one-to-one correspondance (hence the regular expression).

To convert from a supported type (e.g `text`, `json`) to a custom data type and back again, use another pass-through converter:

```
1 | $.ajaxSetup({
2 |   contents: {
3 |     mycustomtype: /mycustomtype/
4 |   },
5 |   converters: {
6 |     "text mycustomtype": true,
7 |     "mycustomtype json": function ( result )
8 |       // do stuff
9 |       return newresult;
10 |   }
11 | }
12 | });
```

The above now allows passing from `text` to `mycustomtype` and then `mycustomtype` to `json`.

## Additional Notes:

Due to browser security restrictions, most "Ajax" requests are subject to the [same origin policy](#); the request can not successfully retrieve data from a different domain, subdomain, or protocol.

Script and JSONP requests are not subject to the same origin policy restrictions.

## Examples:

**Example:** Save some data to the server and notify the user once it's complete.

```
1 $.ajax({
2   type: "POST",
3   url: "some.php",
4   data: { name: "John", location: "Boston" }
5 }).done(function( msg ) {
6   alert( "Data Saved: " + msg );
7 });
```

**Example:** Retrieve the latest version of an HTML page.

```
1 $.ajax({
2   url: "test.html",
3   cache: false
4 }).done(function( html ) {
5   $("#results").append(html);
6 });
```

**Example:** Send an xml document as data to the server. By setting the processData option to false, the automatic conversion of data to strings is prevented.

```
1 var xmlDoc = [create xml document];
2 var xmlRequest = $.ajax({
3   url: "page.php",
4   processData: false,
5   data: xmlDoc
6 });
```

```
7 |  
8 | xmlRequest.done(handleResponse);
```

**Example:** *Send an id as data to the server. save some data to the server and notify the user once it's complete. If the request fails, alert the user.*

```
1 | var menuId = $("ul.nav").first().attr("id");  
2 | var request = $.ajax({  
3 |   url: "script.php",  
4 |   type: "POST",  
5 |   data: {id : menuId},  
6 |   dataType: "html"  
7 | });  
8 |  
9 | request.done(function(msg) {  
10 |   $("#log").html( msg );  
11 | });  
12 |  
13 | request.fail(function(jqXHR, textStatus) {  
14 |   alert( "Request failed: " + textStatus );  
15 | });
```

**Example:** *Load and execute a JavaScript file.*

```
1 | $.ajax({  
2 |   type: "GET",  
3 |   url: "test.js",  
4 |   dataType: "script"  
5 | });
```

**A new version of this book is available!**

---

# jQuery.ajaxPrefilter()

Categories: [Ajax](#) > [Low-Level Interface](#)

`jQuery.ajaxPrefilter( [dataTypes ], handler(options, originalOptions, jqXHR) )` Returns: `undefined`

**Description:** Handle custom Ajax options or modify existing options before each request is sent and before they are processed by `$.ajax()`.

**jQuery.ajaxPrefilter( [dataTypes ], handler(options, originalOptions, jqXHR) )** version added: 1.5

**dataTypes**

Type: [String](#)

An optional string containing one or more space-separated dataTypes

**handler(options, originalOptions, jqXHR)**

Type: [Function\(\)](#)

A handler to set default values for future Ajax requests.

A typical prefilter registration using `$.ajaxPrefilter()` looks like this:

```
1 | $.ajaxPrefilter( function( options, originalOptions, jqXHR ) {
2 |     // Modify options, control originalOptions,
3 | });
```

where:

`options` are the request options

`originalOptions` are the options as provided to the ajax method, unmodified and, thus, without defaults from `ajaxSettings`

`jqXHR` is the jqXHR object of the request

Prefilters are a perfect fit when custom options need to be handled. Given the following code, for example, a call to `$.ajax()` would automatically abort a request to the same URL if the custom

`abortOnRetry` option is set to `true`:

```
1 | var currentRequests = {};  
2 |  
3 | $.ajaxPrefilter(function( options, originalOptions, jqXHR ) {  
4 |     if ( options.abortOnRetry ) {  
5 |         if ( currentRequests[ options.url ] ) {  
6 |             currentRequests[ options.url ].abort();  
7 |         }  
8 |         currentRequests[ options.url ] = jqXHR;  
9 |     }  
10 | });
```

Prefilters can also be used to modify existing options. For example, the following proxies cross-domain requests through `http://mydomain.net/proxy/`:

```
1 | $.ajaxPrefilter( function( options ) {  
2 |     if ( options.crossDomain ) {  
3 |         options.url = "http://mydomain.net/proxy/" + options.url;  
4 |         options.crossDomain = false;  
5 |     }  
6 | });
```

If the optional `dataTypes` argument is supplied, the prefilter will be only be applied to requests with the indicated dataTypes. For example, the following only applies the given prefilter to JSON and script requests:

```
1 | $.ajaxPrefilter( "json script", function( options, originalOptions, jqXHR ) {  
2 |     // Modify options, control originalOptions, control jqXHR.  
3 | });
```

The `$.ajaxPrefilter()` method can also redirect a request to another `dataType` by returning that `dataType`. For example, the following sets a request as "script" if the URL has some specific properties defined in a custom `isActuallyScript()` function:

```
1 | $.ajaxPrefilter(function( options ) {  
2 |     if ( isActuallyScript( options.url ) ) {  
3 |         return "script";  
4 |     }  
5 | });
```

This would ensure not only that the request is considered "script" but also that all the prefilters specifically attached to the script `dataType` would be applied to it.

POWERED BY HERONOTE

A new version of this book is available!

---

# jQuery.ajaxSetup()

Categories: [Ajax](#) > [Low-Level Interface](#)

jQuery.ajaxSetup( options )

Returns:

**Description:** *Set default values for future Ajax requests.*

jQuery.ajaxSetup( options )

version added: [1.1](#)

### options

Type: [PlainObject](#)

A set of key/value pairs that configure the default Ajax request. All options are optional.

For details on the settings available for `$.ajaxSetup()`, see `$.ajax()`.

All subsequent Ajax calls using any function will use the new settings, unless overridden by the individual calls, until the next invocation of `$.ajaxSetup()`.

For example, the following sets a default for the `url` parameter before pinging the server repeatedly:

```
1 | $.ajaxSetup({
2 |   url: 'ping.php'
3 | });
```

Now each time an Ajax request is made, the "ping.php" URL will be used automatically:

```
1 | $.ajax({
2 |   // url not set here; uses ping.php
3 |   data: { 'name': 'Dan' }
4 | });
```

*Note: Global callback functions should be*

*set with their respective global Ajax event handler methods—`.ajaxStart()`, `.ajaxStop()`, `.ajaxComplete()`, `.ajaxError()`, `.ajaxSuccess()`, `.ajaxSend()`—rather than within the `options` object for `$.ajaxSetup()`.*

## Example:

*Sets the defaults for Ajax requests to the url "/xmlhttp/" disables global handlers and uses POST instead of GET The following Ajax requests then sends some data without having to set anything else.*

```
1 $.ajaxSetup({
2   url: "/xmlhttp/",
3   global: false,
4   type: "POST"
5
6 });
7 $.ajax({ data: myData });
```

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# jQuery.ajaxTransport()

Categories: [Ajax](#) > [Low-Level Interface](#)

jQuery.ajaxTransport( dataType, handler(options, originalOptions, jqXHR) ) Returns: *undefined*

**Description:** *Creates an object that handles the actual transmission of Ajax data.*

jQuery.ajaxTransport( dataType, handler(options, originalOptions, jqXHR) ) version added: 1.5

**dataType**

Type: [String](#)

A string identifying the data type to use

**handler(options, originalOptions, jqXHR)**

Type: [Function\(\)](#)

A handler to return the new transport object to use with the data type provided in the first argument.

A transport is an object that provides two methods, `send` and `abort`, that are used internally by `$.ajax()` to issue requests. A transport is the most advanced way to enhance `$.ajax()` and should be used only as a last resort when prefilters and converters are insufficient.

Since each request requires its own transport object instance, transports cannot be registered directly. Therefore, you should provide a function that returns a transport instead.

Transports factories are registered using `$.ajaxTransport()`. A typical registration looks like this:

```
1 | $.ajaxTransport( function( options, originalOptions, jqXHR ) {
2 |     if( /* transportCanHandleRequest */ ) {
3 |         return {
4 |             send: function( headers, completeCallback, dataType, dataFilter ) {
5 |                 /* send code */
6 |             },
7 |             abort: function() {
```

```
8 | /* abort code */  
9 |     }  
10 | };  
11 | }  
12 | });
```

where:

`options` are the request options

`originalOptions` are the options as provided to the ajax method, unmodified and, thus, without defaults from `ajaxSettings`

`jqXHR` is the jqXHR object of the request

`headers` is an object of (key-value) request headers that the transport can transmit if it supports it

`completeCallback` is the callback used to notify ajax of the completion of the request

`completeCallback` has the following signature:

```
1 | function( status, textStatus, responses, headers
```

where:

`status` is the HTTP status code of the response, like 200 for a typical success, or 404 for when the resource is not found.

`statusText` is the `statusText` of the response.

`responses` (Optional) is An object containing `dataType/value` that contains the response in all the formats the transport could provide (for instance, a native `XMLHttpRequest` object would set `responses` to `{ xml: XMLData, text: textData }` for a response that is an XML document)

`headers` (Optional) is a string containing all the response headers if the transport has access to them (akin to what `XMLHttpRequest.getAllResponseHeaders()` would provide).

Just like prefilters, a transport's factory function can be attached to a specific dataType:

```
1 | $.ajaxTransport( "script", function( options,  
2 |   /* Will only be called for script requests  
3 | });
```

The following example shows how a minimal image transport could be implemented:

```
1 | $.ajaxTransport( "image", function( s ) {  
2 |  
3 |   if ( s.type === "GET" && s.async ) {  
4 |  
5 |     var image;  
6 |  
7 |     return {  
8 |  
9 |       send: function( _ , callback ) {  
10 |  
11 |         image = new Image();  
12 |  
13 |         function done( status ) {  
14 |           if ( image ) {  
15 |             var textStatus = ( status == 200  
16 |               tmp = image;  
17 |               image = image.onreadystatechange  
18 |               callback( status, textStatus, { :  
19 |             }  
20 |           }  
21 |         }  
22 |         image.onreadystatechange = image.onload  
23 |         done( 200 );  
24 |       };  
25 |       image.onerror = function() {
```

```

26         done( 404 );
27     };
28
29     image.src = s.url;
30 },
31
32     abort: function() {
33         if ( image ) {
34             image.onreadystatechange = image.oner
35         }
36     }
37 };
38 }
39 });

```

## Handling Custom Data Types

The jQuery Ajax implementation comes with a set of standard dataTypes, such as text, json, xml, and html.

Use the `converters` option in `$.ajaxSetup()` to augment or modify the data type conversion strategies used by `$.ajax()`.

The unminified jQuery source itself includes a list of default converters, which effectively illustrates how they can be used:

```

1 // List of data converters
2 // 1) key format is "source_type destination_
3 //   (a single space in-between)
4 // 2) the catchall symbol "*" can be used for
5 converters: {
6
7     // Convert anything to text
8     "* text": window.String,
9
10    // Text to html (true = no transformation)
11    "text html": true,

```

```
12 |  
13 | // Evaluate text as a json expression  
14 | "text json": jQuery.parseJSON,  
15 |  
16 | // Parse text as xml  
17 | "text xml": jQuery.parseXML  
18 | }
```

When you specify a `converters` option globally in `$.ajaxSetup()` or per call in `$.ajax()`, the object will map onto the default converters, overwriting those you specify and leaving the others intact.

For example, the jQuery source uses `$.ajaxSetup()` to add a converter for "text script":

```
1 | jQuery.ajaxSetup({  
2 |   accepts: {  
3 |     script: "text/javascript, application/javascript",  
4 |   },  
5 |   contents: {  
6 |     script: /javascript/,  
7 |   },  
8 |   converters: {  
9 |     "text script": jQuery.globalEval,  
10 |   }  
11 | });
```

**A new version of this book is available!**

---

# jQuery.boxModel

Categories: [Utilities](#)

jQuery.boxModel

Returns: [Boolean](#)  
version deprecated: [1.3](#)

**Description:** **Deprecated in jQuery 1.3 (see [jQuery.support](#)).** *States if the current page, in the user's browser, is being rendered using the [W3C CSS Box Model](#).*

jQuery.boxModel

version added: [1.0](#)

## Examples:

**Example:** *Returns the box model for the iframe.*

```
1 | <!DOCTYPE html>
2 | <html>
3 | <head>
4 |   <style>
5 |
6 |   p { color:blue; margin:20px; }
7 |   span { color:red; }
8 | </style>
9 |   <script src="http://code.jquery.com/jquery-
10| </head>
11| <body>
12|   <p>
13|   </p>
14| <script>
15| $( "p" ).html( "The box model for this iframe is
16|               jQuery.boxModel + "</span>" );
17| </script>
18|
19| </body>
20| </html>
```

Demo

**Example:** *Returns false if the page is in Quirks Mode in Internet Explorer*

```
1 | $.boxModel
```

**Result:**

1 | false

POWERED BYHERONOTE

A new version of this book is available!

---

# jQuery.browser

Categories: [Properties](#) > [Properties of the Global jQuery Object](#) | [Utilities](#)

## Contents:

[jQuery.browser](#)

[jQuery.browser.version](#)

jQuery.browser

Returns: PlainObject

version deprecated: 1.3, removed: 1.9

**Description:** *Contains flags for the userAgent, read from navigator.userAgent. We recommend against using this property: please try to use feature detection instead (see jQuery.support). jQuery.browser may be moved to a plugin in a future release of jQuery.*

jQuery.browser

version added: 1.0

The `$.browser` property provides information about the web browser that is accessing the page, as reported by the browser itself. It contains flags for each of the four most prevalent browser classes (Internet Explorer, Mozilla, Webkit, and Opera) as well as version information.

Available flags are:

- webkit (as of jQuery 1.4)
- safari (deprecated)
- opera
- msie
- mozilla

This property is available immediately. It is therefore safe to use it to determine whether or not to call `$(document).ready()`. The `$.browser` property is deprecated in jQuery 1.3, and its functionality may be moved to a team-supported plugin in a future release of jQuery.

Because `$.browser` uses `navigator.userAgent` to determine the platform, it is vulnerable to spoofing by the user or misrepresentation by the browser itself. It is always best to avoid browser-specific code entirely where possible. The `$.support` property is available for detection of support for particular features rather than relying on `$.browser`.



## Examples:

**Example:** *Show the browser info.*

```
1 | jQuery.each(jQuery.browser, function(i, val) {  
2 |     $("<div>" + i + " : <span>" + val + "</span>"  
3 |     .appendTo( document.body );  
4 | });
```

**Example:** *Returns true if the current useragent is some version of Microsoft's Internet Explorer.*

```
1 | $.browser.msie;
```

**Example:** *Alerts "this is WebKit!" only for WebKit browsers*

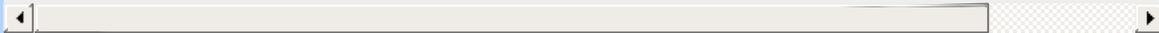
```
1 | if ($.browser.webkit) {  
2 |     alert( "this is webkit!" );  
3 | }
```

**Example:** *Alerts "Do stuff for Firefox 3" only for Firefox 3 browsers.*

```
1 | var ua = $.browser;  
2 | if ( ua.mozilla && ua.version.slice(0,3) ==  
3 |     alert( "Do stuff for firefox 3" );  
4 | }
```

**Example** Set a CSS property that's specific to a particular browser.

```
1 | if ( $.browser.msie ) {  
2 |     $( "#div ul li" ).css( "display", "inline" );  
3 | } else {  
4 |     $( "#div ul li" ).css( "display", "inline-tab.  
5 | }
```



`jQuery.browser.version`

Returns: [String](#)

version deprecated: [1.3](#), removed: [1.9](#)

**Description:** *The version number of the rendering engine for the user's browser.*

`jQuery.browser.version`

version added: [1.1.3](#)

Here are some typical results:

Internet Explorer: 6.0, 7.0, 8.0

Mozilla/Firefox/Flock/Camino: 1.7.12, 1.8.1.3, 1.9

Opera: 10.06, 11.01

Safari/Webkit: 312.8, 418.9

Note that IE8 claims to be 7 in Compatibility View.

## Examples:

**Example:** Returns the version number of the rendering engine used by the user's current browser. For example, Firefox 4 returns 2.0 (the version of the Gecko rendering engine it utilizes).

```
1 | $("p").html( "The version number of the render.  
2 |             $.browser.version + "</span>"
```

**Example:** Alerts the version of IE's rendering engine that is being used:

```
1 | if ( $.browser.msie ) {  
2 |     alert( $.browser.version );  
3 | }
```

**Example:** Often you only care about the "major number" the whole number which you can get by using JavaScript's built-in `parseInt()` function:

```
1 | if ( $.browser.msie ) {  
2 |     alert( parseInt($.browser.version, 10) );  
3 | }
```

**A new version of this book is available!**

---

# jQuery.Callbacks()

Categories: [Callbacks Object](#)

`jQuery.Callbacks( flags )`

Returns: [Callbacks](#)

**Description:** *A multi-purpose callbacks list object that provides a powerful way to manage callback lists.*

`jQuery.Callbacks( flags )`

version added: [1.7](#)

**flags**

Type: [String](#)

An optional list of space-separated flags that change how the callback list behaves.

The `$.Callbacks()` function is internally used to provide the base functionality behind the jQuery `$.ajax()` and `$.Deferred()` components. It can be used as a similar base to define functionality for new components.

`$.Callbacks()` supports a number of methods including `callbacks.add()`, `callbacks.remove()`, `callbacks.fire()` and `callbacks.disable()`.

## Getting started

The following are two sample methods named `fn1` and `fn2`:

```
1 function fn1( value ) {  
2     console.log( value );  
3 }  
4  
5 function fn2( value ) {  
6     fn1("fn2 says: " + value);  
7     return false;  
8 }
```

These can be added as callbacks to a `$.Callbacks` list and invoked as follows:

```

1 | var callbacks = $.Callbacks();
2 | callbacks.add( fn1 );
3 |
4 | // outputs: foo!
5 | callbacks.fire( "foo!" );
6 |
7 | callbacks.add( fn2 );
8 |
9 | // outputs: bar!, fn2 says: bar!
10 | callbacks.fire( "bar!" );

```

The result of this is that it becomes simple to construct complex lists of callbacks where input values can be passed through to as many functions as needed with ease.

Two specific methods were being used above: `.add()` and `.fire()`. The `.add()` method supports adding new callbacks to the callback list, while the `.fire()` method executes the added functions and provides a way to pass arguments to be processed by the callbacks in the same list.

Another method supported by `$.Callbacks` is `.remove()`, which has the ability to remove a particular callback from the callback list. Here's a practical example of `.remove()` being used:

```

1 | var callbacks = $.Callbacks();
2 | callbacks.add( fn1 );
3 |
4 | // outputs: foo!
5 | callbacks.fire( "foo!" );
6 |
7 | callbacks.add( fn2 );
8 |
9 | // outputs: bar!, fn2 says: bar!
10 | callbacks.fire( "bar!" );
11 |
12 | callbacks.remove( fn2 );

```

```
13 |  
14 | // only outputs foobar, as fn2 has been removed.  
15 | callbacks.fire( "foobar" );
```

## Supported Flags

The `flags` argument is an optional argument to `$.Callbacks()`, structured as a list of space-separated strings that change how the callback list behaves (eg. `$.Callbacks( "unique stopOnFalse" )`).

## Possible flags:

`once`: Ensures the callback list can only be fired once (like a Deferred).

`memory`: Keeps track of previous values and will call any callback added after the list has been fired right away with the latest "memorized" values (like a Deferred).

`unique`: Ensures a callback can only be added once (so there are no duplicates in the list).

`stopOnFalse`: Interrupts callings when a callback returns false.

By default a callback list will act like an event callback list and can be "fired" multiple times.

For examples of how `flags` should ideally be used, see below:

```
$.Callbacks( "once" ):
```

```
1  var callbacks = $.Callbacks( "once" );
2  callbacks.add( fn1 );
3  callbacks.fire( "foo" );
4  callbacks.add( fn2 );
5  callbacks.fire( "bar" );
6  callbacks.remove( fn2 );
7  callbacks.fire( "foobar" );
8
9  /*
10 output:
11 foo
12 */
```

```
$.Callbacks( "memory" ) :
```

```
1  var callbacks = $.Callbacks( "memory" );
2  callbacks.add( fn1 );
3  callbacks.fire( "foo" );
4  callbacks.add( fn2 );
5  callbacks.fire( "bar" );
6  callbacks.remove( fn2 );
7  callbacks.fire( "foobar" );
8
9  /*
10 output:
11 foo
12 fn2 says:foo
13 bar
14 fn2 says:bar
15 foobar
16 */
```

```
$.Callbacks( "unique" ):
```

```
1  var callbacks = $.Callbacks( "unique" );
2  callbacks.add( fn1 );
3  callbacks.fire( "foo" );
4  callbacks.add( fn1 ); // repeat addition
5  callbacks.add( fn2 );
6  callbacks.fire( "bar" );
7  callbacks.remove( fn2 );
8  callbacks.fire( "foobar" );
9
10 /*
11  output:
12  foo
13  bar
14  fn2 says:bar
15  foobar
16  */
```

```
$.Callbacks( "stopOnFalse" ):
```

```
1  function fn1( value ){
2      console.log( value );
3      return false;
4  }
5
6  function fn2( value ){
7      fn1( "fn2 says: " + value );
8      return false;
9  }
10
11  var callbacks = $.Callbacks( "stopOnFalse" );
12  callbacks.add( fn1 );
13  callbacks.fire( "foo" );
14  callbacks.add( fn2 );
15  callbacks.fire( "bar" );
16  callbacks.remove( fn2 );
17  callbacks.fire( "foobar" );
18
19  /*
20  output:
21  foo
22  bar
23  foobar
24  */
```

Because `$.Callbacks()` supports a list of flags rather than just one, setting several flags has a cumulative effect similar to "&&". This means it's possible to combine flags to create callback lists that, say, both are *unique* and *ensure* if list was already fired, adding more callbacks will have it called with the latest fired value (i.e.

```
$.Callbacks("unique memory").
```



```
$.Callbacks( 'unique memory' ):
```

```
1  function fn1( value ) {
2      console.log( value );
3      return false;
4  }
5
6  function fn2( value ) {
7      fn1( "fn2 says: " + value );
8      return false;
9  }
10
11  var callbacks = $.Callbacks( "unique memory"
12  callbacks.add( fn1 );
13  callbacks.fire( "foo" );
14  callbacks.add( fn1 ); // repeat addition
15  callbacks.add( fn2 );
16  callbacks.fire( "bar" );
17  callbacks.add( fn2 );
18  callbacks.fire( "baz" );
19  callbacks.remove( fn2 );
20  callbacks.fire( "foobar" );
21
22  /*
23  output:
24  foo
25  fn2 says:foo
26  bar
27  fn2 says:bar
28  baz
29  fn2 says:baz
30  foobar
31  */
```

Flag combinations with `$.Callbacks()` are internally in jQuery for the `.done()` and `.fail()` functions on a Deferred — both of which use `$.Callbacks('memory once')`.

The methods of `$.Callbacks` can also be detached, should there be a need to define short-hand versions for convenience:

```
1 | var callbacks = $.Callbacks(),
2 |   add = callbacks.add,
3 |   remove = callbacks.remove,
4 |   fire = callbacks.fire;
5 |
6 | add( fn1 );
7 | fire( "hello world" );
8 | remove( fn1 );
```

## \$.Callbacks, \$.Deferred and Pub/Sub

The general idea behind pub/sub (Publish/Subscribe, or, the Observer pattern) is the promotion of loose coupling in applications. Rather than single objects calling on the methods of other objects, an object instead subscribes to a specific task or activity of another object and is notified when it occurs. Observers are also called Subscribers, and we refer to the object being observed as the Publisher (or the subject). Publishers notify subscribers when events occur.

To demonstrate the component-creation capabilities of `$.Callbacks()`, it's possible to implement a Pub/Sub system using only callback lists. Using `$.Callbacks` as a topics queue, a system for publishing and subscribing to topics can be implemented as follows:

```
1 | var topics = {};
2 |
3 | jQuery.Topic = function( id ) {
4 |   var callbacks,
```

```

5      method,
6      topic = id && topics[ id ];
7
8      if ( !topic ) {
9          callbacks = jQuery.Callbacks();
10         topic = {
11             publish: callbacks.fire,
12             subscribe: callbacks.add,
13             unsubscribe: callbacks.remove
14         };
15         if ( id ) {
16             topics[ id ] = topic;
17         }
18     }
19     return topic;
20 };

```

This can then be used by parts of your application to publish and subscribe to events of interest quite easily:

```

1  // Subscribers
2  $.Topic( "mailArrived" ).subscribe( fn1 );
3  $.Topic( "mailArrived" ).subscribe( fn2 );
4  $.Topic( "mailSent" ).subscribe( fn1 );
5
6  // Publisher
7  $.Topic( "mailArrived" ).publish( "hello worl
8  $.Topic( "mailSent" ).publish( "woo! mail!"
9
10 // Here, "hello world!" gets pushed to fn1 at
11 // when the "mailArrived" notification is pul
12 // with "woo! mail!" also being pushed to fn:
13 // the "mailSent" notification is published.
14
15 /*
16 output:

```

```
17 | hello world!  
18 | fn2 says: hello world!  
19 | woo! mail!  
20 | */
```

While this is useful, the implementation can be taken further. Using `$.Deferreds`, it's possible to ensure publishers only publish notifications for subscribers once particular tasks have been completed (resolved). See the below code sample for some further comments on how this could be used in practice:

```
1 | // subscribe to the mailArrived notification  
2 | $.Topic( "mailArrived" ).subscribe( fn1 );  
3 |  
4 | // create a new instance of Deferreds  
5 | var dfd = $.Deferred();  
6 |  
7 | // define a new topic (without directly publ:  
8 | var topic = $.Topic( "mailArrived" );  
9 |  
10 | // when the deferred has been resolved, publ:  
11 | // notification to subscribers  
12 | dfd.done( topic.publish );  
13 |  
14 | // Here the Deferred is being resolved with a  
15 | // that will be passed back to subscribers. :  
16 | // easily integrate this into a more complex  
17 | // (eg. waiting on an ajax call to complete)  
18 | // messages are only published once the task  
19 | // finished.  
20 | dfd.resolve( "it's been published!" );
```

POWERED BY HERONOTE

A new version of this book is available!

---

# jQuery.contains()

Categories: [Utilities](#)

`jQuery.contains( container, contained )`

Returns: [Boolean](#)

**Description:** *Check to see if a DOM element is a descendant of another DOM element.*

**jQuery.contains( container, contained )** version added: [1.4](#)  
)

**container**

Type: [Element](#)

The DOM element that may contain the other element.

**contained**

Type: [Element](#)

The DOM element that may be contained by (a descendant of) the other element.

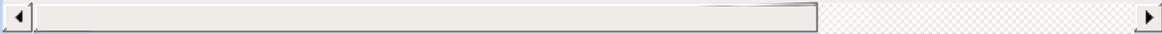
The `$.contains()` method returns `true` if the DOM element provided by the second argument is a descendant of the DOM element provided by the first argument, whether it is a direct child or nested more deeply. Otherwise, it returns `false`. Only *element* nodes are supported; if the second argument is a text or comment node, `$.contains()` will return `false`.

**Note:** *The first argument **must** be a DOM element, not a jQuery object or plain JavaScript object.*

## Example:

*Check if an element is a descendant of another.*

```
1 | $.contains( document.documentElement, document  
2 | $.contains( document.body, document.documentEl
```



POWERED BY **HERONOTE**

A new version of this book is available!

---

# jQuery.cssHooks

Categories: [CSS](#)

## jQuery.cssHooks

Returns: [Object](#)

**Description:** Hook directly into jQuery to override how particular CSS properties are retrieved or set, normalize CSS property naming, or create custom properties.

## jQuery.cssHooks

version added: [1.4.3](#)

The `$.cssHooks` object provides a way to define functions for getting and setting particular CSS values. It can also be used to create new `cssHooks` for normalizing CSS3 features such as box shadows and gradients.

For example, some versions of Webkit-based browsers require `-webkit-border-radius` to set the `border-radius` on an element, while earlier Firefox versions require `-moz-border-radius`. A `css` hook can normalize these vendor-prefixed properties to let `.css()` accept a single, standard property name (`border-radius`, or with DOM property syntax, `borderRadius`).

In addition to providing fine-grained control over how specific style properties are handled, `$.cssHooks` also extends the set of properties available to the `.animate()` method.

Defining a new `css` hook is straight-forward. The skeleton template below can serve as a guide to creating your own.

```
1 | (function($) {
2 |     // first, check to see if cssHooks are supported
3 |     if ( !$.cssHooks ) {
4 |         // if not, output an error message
5 |         throw("jQuery 1.4.3 or above is required");
6 |         return;
7 |     }
8 |
9 |     // Wrap in a document ready call, because $.cssHooks
10 |    // cssHooks at this time and will blow away
```

```

11 // if they exist.
12 $(function () {
13     $.cssHooks["someCSSProp"] = {
14         get: function( elem, computed, extra ) {
15             // handle getting the CSS property
16         },
17         set: function( elem, value ) {
18             // handle setting the CSS value
19         }
20     };
21 });
22 })(jQuery);

```

## Feature Testing

Before normalizing a vendor-specific CSS property, first determine whether the browser supports the standard property or a vendor-prefixed variation. For example, to check for support of the `border-radius` property, see if any variation is a member of a temporary element's `style` object.

```

1 (function($) {
2     function styleSupport( prop ) {
3         var vendorProp, supportedProp,
4
5         // capitalize first character of the
6         capProp = prop.charAt(0).toUpperCase(),
7         prefixes = [ "Moz", "Webkit", "O", "r
8         div = document.createElement( "div"
9
10        if ( prop in div.style ) {
11
12            // browser supports standard CSS propert
13            supportedProp = prop;
14        } else {
15

```

```

16 // otherwise test support for vendor-prefixed prop
17 for ( var i = 0; i < prefixes.length; i++ ) {
18     vendorProp = prefixes[i] + capProp;
19     if ( vendorProp in div.style ) {
20         supportedProp = vendorProp;
21         break;
22     }
23 }
24 }
25
26 // avoid memory leak in IE
27 div = null;
28
29 // add property to $.support so it can be accessed
30 $.support[ prop ] = supportedProp;
31
32 return supportedProp;
33 }
34
35 // call the function, e.g. testing for "border-radius"
36 styleSupport( "borderRadius" );
37 })(jQuery);

```

## Defining a complete css hook

To define a complete css hook, combine the support test with a filled-out version of the skeleton template provided in the first example:

```

1 (function($) {
2     if ( !$ .cssHooks ) {
3         throw("jQuery 1.4.3+ is needed for this plugin");
4         return;
5     }
6
7     function styleSupport( prop ) {

```

```

8     var vendorProp, supportedProp,
9         capProp = prop.charAt(0).toUpperCase() + prop.s
10    prefixes = [ "Moz", "Webkit", "O", "ms"
11    div = document.createElement( "div" );
12
13    if ( prop in div.style ) {
14        supportedProp = prop;
15    } else {
16        for ( var i = 0; i < prefixes.length; i++ ) {
17            vendorProp = prefixes[i] + capProp;
18            if ( vendorProp in div.style ) {
19                supportedProp = vendorProp;
20                break;
21            }
22        }
23    }
24
25    div = null;
26    $.support[ prop ] = supportedProp
27    return supportedProp;
28 }
29
30 var borderRadius = styleSupport( "borderRadius"
31
32 // Set cssHooks only for browsers that
33 // support a vendor-prefixed border radius
34 if ( borderRadius && borderRadius !== "borderRadius"
35     $.cssHooks.borderRadius = {
36     get: function( elem, computed, extra ) {
37         return $.css( elem, borderRadius );
38     },
39     set: function( elem, value ) {
40         elem.style[ borderRadius ] = value;
41     }
42     };
43 }
44 })(jQuery);

```

You can then set the border radius in a supported browser using either the DOM (camelCased) style or the CSS (hyphenated) style:

```
1 | $( "#element" ).css( "borderRadius", "10px" );  
2 | $( "#another" ).css( "border-radius", "20px" );
```

If the browser lacks support for any form of the CSS property, vendor-prefixed or not, the style is not applied to the element. However, if the browser supports a proprietary alternative, it can be applied to the `cssHooks` instead.

```
1 | (function($) {  
2 |     // feature test for support of a CSS property  
3 |     // and a proprietary alternative  
4 |     // ...  
5 |  
6 |  
7 |     if ( $.support.someCSSProp && $.support.someProprietaryAlternative )  
8 |  
9 |         // Set cssHooks for browsers that  
10 |         // support only a vendor-prefixed someCSSProp  
11 |         $.cssHooks.someCSSProp = {  
12 |             get: function( elem, computed, extra )  
13 |                 return $.css( elem, $.support.someCSSProp, extra );  
14 |             },  
15 |             set: function( elem, value ) {  
16 |                 elem.style[ $.support.someCSSProp ] = value;  
17 |             }  
18 |         };  
19 |     } else if ( supportsProprietaryAlternative )  
20 |         $.cssHooks.someCSSProp = {  
21 |             get: function( elem, computed, extra )  
22 |                 // Handle crazy conversion from the proprietary alternative  
23 |                 return $.css( elem, $.support.someCSSProp, extra );  
24 |             },  
25 |             set: function( elem, value ) {  
26 |                 elem.style[ $.support.someProprietaryAlternative ] = value;  
27 |             }  
28 |         }  
29 | } );
```

```
24 |         set: function( elem, value ) {
25 |             // Handle crazy conversion to the proprietary a
26 |         }
27 |     }
28 | }
29 |
30 | })(jQuery);
```

## Special units

By default, jQuery adds a "px" unit to the values passed to the `.css()` method. This behavior can be prevented by adding the property to the `jQuery.cssNumber` object

```
1 | $.cssNumber["someCSSProp"] = true;
```

## Animating with cssHooks

A css hook can also hook into jQuery's animation mechanism by adding a property to the `jQuery.fx.step` object:

```
1 | $.fx.step["someCSSProp"] = function(fx){
2 |     $.cssHooks["someCSSProp"].set( fx.elem, fx.r
3 | };
```

Note that this works best for simple numeric-value animations. More custom code may be required depending on the CSS property, the type of value it returns, and the animation's complexity.

**A new version of this book is available!**

---

# jQuery.data()

Categories: [Data](#) | [Utilities](#)

Store arbitrary data associated with the specified element and/or return the value that was set.

## Contents:

[jQuery.data\( element, key, value \)](#)

jQuery.data( element, key, value )

[jQuery.data\( element, key \)](#)

jQuery.data( element, key )

jQuery.data( element )

---

`jQuery.data( element, key, value )`

Returns: [Object](#)

**Description:** Store arbitrary data associated with the specified element. Returns the value that was set.

`jQuery.data( element, key, value )`

version added: [1.2.3](#)

**element**

Type: [Element](#)

The DOM element to associate with the data.

**key**

Type: [String](#)

A string naming the piece of data to set.

**value**

Type: [Object](#)

The new data value.

**Note:** This is a low-level method; a more convenient `.data()` is also available.

The `jQuery.data()` method allows us to attach data of any type to DOM elements in a way that is safe from circular references and therefore free from memory leaks. jQuery ensures that the data is removed when DOM elements are removed via jQuery methods, and when the user leaves the page. We can set several distinct values for a single element and retrieve them later:

```
1 | jQuery.data( document.body, 'foo', 52 );  
2 | jQuery.data( document.body, 'bar', 'test' );
```

*Note:* this method currently does not provide cross-platform support for setting data on XML documents, as Internet Explorer does not allow data to be attached via expando properties.

## Example:

*Store then retrieve a value from the div element.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5      div { color:blue; }
6      span { color:red; }
7    </style>
8    <script src="http://code.jquery.com/jquery-
9  </head>
10 <body>
11   <div>
12     The values stored were
13     <span></span>
14     and
15     <span></span>
16   </div>
17   <script>var div = $("div")[0];
18     jQuery.data(div, "test", { first: 16, las
19     $("span:first").text(jQuery.data(div, "te
20     $("span:last").text(jQuery.data(div, "tes
21
22 </body>
23 </html>
```

Demo

`jQuery.data( element, key )`

Returns: [Object](#)

**Description:** Returns value at named data store for the element, as set by `jQuery.data(element, name, value)`, or the full data store for the element.

`jQuery.data( element, key )`

version added: [1.2.3](#)

**element**

Type: [Element](#)

The DOM element to query for the data.

**key**

Type: [String](#)

Name of the data stored.

`jQuery.data( element )`

version added: [1.4](#)

**element**

Type: [Element](#)

The DOM element to query for the data.

**Note:** This is a low-level method; a more convenient `.data()` is also available.

**Regarding HTML5 data-\* attributes:** This low-level method does NOT retrieve the `data-*` attributes unless the more convenient `.data()` method has already retrieved them.

The `jQuery.data()` method allows us to attach data of any type to DOM elements in a way that is safe from circular references and therefore from memory leaks. We can retrieve several distinct values for a single element one at a time, or as a set:

```
1 | alert(jQuery.data( document.body, 'foo' ));
2 | alert(jQuery.data( document.body ));
```

The above lines alert the data values that were set on the `body`

element. If nothing was set on that element, an empty string is returned.

Calling `jQuery.data(element)` retrieves all of the element's associated values as a JavaScript object. Note that jQuery itself uses this method to store data for internal use, such as event handlers, so do not assume that it contains only data that your own code has stored.

*Note:* this method currently does not provide cross-platform support for setting data on XML documents, as Internet Explorer does not allow data to be attached via expando properties.

## Example:

*Get the data named "blah" stored at for an element.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5   div { margin:5px; background:yellow; }
6   button { margin:5px; font-size:14px; }
7   p { margin:5px; color:blue; }
8   span { color:red; }
9   </style>
10  <script src="http://code.jquery.com/jquery-
11 </head>
12 <body>
13   <div>A div</div>
14   <button>Get "blah" from the div</button>
15   <button>Set "blah" to "hello"</button>
16
17   <button>Set "blah" to 86</button>
18   <button>Remove "blah" from the div</button>
19   <p>The "blah" value of this div is <span>?</span>
20 </script>
21 $( "button" ).click( function( e ) {
22   var value, div = $( "div" )[ 0 ];
23
24   switch ( $( "button" ).index( this ) ) {
25     case 0 :
26       value = jQuery.data( div, "blah" );
27       break;
28     case 1 :
29       jQuery.data( div, "blah", "hello" );
30       value = "Stored!";
31     break;
```

```
32     case 2 :
33         jQuery.data(div, "blah", 86);
34         value = "Stored!";
35         break;
36     case 3 :
37         jQuery.removeData(div, "blah");
38         value = "Removed!";
39         break;
40     }
41
42     $("span").text("" + value);
43 });
44
45 </script>
46
47 </body>
48 </html>
```

Demo

POWERED BY HERONOTE

A new version of this book is available!

---

# jQuery.Deferred()

Categories: [Deferred Object](#)

`jQuery.Deferred( [beforeStart ] )`

Returns: [Deferred](#)

**Description:** *A constructor function that returns a chainable utility object with methods to register multiple callbacks into callback queues, invoke callback queues, and relay the success or failure state of any synchronous or asynchronous function.*

`jQuery.Deferred( [beforeStart ] )`

version added: [1.5](#)

#### **beforeStart**

Type: [Function](#)( [Deferred](#) deferred )

A function that is called just before the constructor returns.

The `jQuery.Deferred()` constructor creates a new `Deferred` object. The `new` operator is optional.

The `jQuery.Deferred` method can be passed an optional function, which is called just before the constructor returns and is passed the constructed `deferred` object as both the `this` object and as the first argument to the function. The called function can attach callbacks using `deferred.then()`, for example.

A `Deferred` object starts in the *pending* state. Any callbacks added to the object with `deferred.then()`, `deferred.always()`, `deferred.done()`, or `deferred.fail()` are queued to be executed later. Calling `deferred.resolve()` or `deferred.resolveWith()` transitions the `Deferred` into the *resolved* state and immediately executes any `doneCallbacks` that are set. Calling `deferred.reject()` or `deferred.rejectWith()` transitions the `Deferred` into the *rejected* state and immediately executes any `failCallbacks` that are set. Once the object has entered the resolved or rejected state, it stays in that state. Callbacks can still be added to the resolved or rejected `Deferred` — they will execute immediately.

### **Enhanced Callbacks with jQuery Deferred**

In JavaScript it is common to invoke functions that optionally accept callbacks that are called within that function. For example, in

versions prior to jQuery 1.5, asynchronous processes such as `jQuery.ajax()` accept callbacks to be invoked some time in the near-future upon success, error, and completion of the ajax request.

`jQuery.Deferred()` introduces several enhancements to the way callbacks are managed and invoked. In particular, `jQuery.Deferred()` provides flexible ways to provide multiple callbacks, and these callbacks can be invoked regardless of whether the original callback dispatch has already occurred. jQuery Deferred is based on the [CommonJS Promises/A](#) design.

One model for understanding Deferred is to think of it as a chain-aware function wrapper. The `deferred.then()`, `deferred.always()`, `deferred.done()`, and `deferred.fail()` methods specify the functions to be called and the `deferred.resolve(args)` or `deferred.reject(args)` methods "call" the functions with the arguments you supply. Once the Deferred has been resolved or rejected it stays in that state; a second call to `deferred.resolve()`, for example, is ignored. If more functions are added by `deferred.then()`, for example, after the Deferred is resolved, they are called immediately with the arguments previously provided.

In most cases where a jQuery API call returns a Deferred or Deferred-compatible object, such as `jQuery.ajax()` or `jQuery.when()`, you will only want to use the `deferred.then()`, `deferred.done()`, and `deferred.fail()` methods to add callbacks to the Deferred's queues. The internals of the API call or code that created the Deferred will invoke `deferred.resolve()` or `deferred.reject()` on the deferred at some point, causing the appropriate callbacks to run.

**POWERED BYHERONOTE**

A new version of this book is available!

---

# jQuery.dequeue()

Categories: [Data](#) | [Utilities](#)

`jQuery.dequeue( element [, queueName ] )`

Returns: `undefined`

**Description:** *Execute the next function on the queue for the matched element.*

`jQuery.dequeue( element [, queueName ] )`

version added: `1.3`

**element**

Type: `Element`

A DOM element from which to remove and execute a queued function.

**queueName**

Type: `String`

A string containing the name of the queue. Defaults to `fx`, the standard effects queue.

**Note:** This is a low-level method, you should probably use `.dequeue()` instead.

When `jQuery.dequeue()` is called, the next function on the queue is removed from the queue, and then executed. This function should in turn (directly or indirectly) cause `jQuery.dequeue()` to be called, so that the sequence can continue.

## Example:

*Use `jQuery.dequeue()` to end a custom queue function which allows the queue to keep going.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>div {
5     margin:3px; width:50px; position:absolute;
6     height:50px; left:10px; top:30px;
7     background-color:yellow;
8   }
9   div.red { background-color:red; } </style>
10  <script src="http://code.jquery.com/jquery-
11 </head>
12 <body>
13   <button>Start</button> <div></div>
14   <script>$("button").click(function () {
15     $("div").animate({left:'+=200px'}, 2000);
16     $("div").animate({top:'0px'}, 600);
17     $("div").queue(function () {
18       $(this).toggleClass("red");
19       $.dequeue( this );
20     });
21     $("div").animate({left:'10px', top:'30px'},
22   });</script>
23
24 </body>
25 </html>
```

Demo

POWERED BY HERONOTE

A new version of this book is available!

---

# jQuery.each()

Categories: [Utilities](#)

jQuery.each( collection, callback(indexInArray, valueOfElement) ) Returns: [Object](#)

**Description:** A generic iterator function, which can be used to seamlessly iterate over both objects and arrays. Arrays and array-like objects with a length property (such as a function's arguments object) are iterated by numeric index, from 0 to length-1. Other objects are iterated via their named properties.

jQuery.each( collection, callback(indexInArray, valueOfElement) ) version added: [1.0](#)

**collection**

Type: [Object](#)

The object or array to iterate over.

**callback(indexInArray, valueOfElement)**

Type: [Function\(\)](#)

The function that will be executed on every object.

The `$.each()` function is not the same as `$(selector).each()`, which is used to iterate, exclusively, over a jQuery object. The `$.each()` function can be used to iterate over any collection, whether it is an object or an array. In the case of an array, the callback is passed an array index and a corresponding array value each time. (The value can also be accessed through the `this` keyword, but Javascript will always wrap the `this` value as an `object` even if it is a simple string or number value.) The method returns its first argument, the object that was iterated.

```
1 | $.each([52, 97], function(index, value) {  
2 |     alert(index + ': ' + value);  
3 | });
```

This produces two messages:

0: 52 1: 97

If an object is used as the collection, the callback is passed a key-value pair each time:

```
1 | var obj = {  
2 |   "flammable": "inflammable",  
3 |   "duh": "no duh"  
4 | };  
5 | $.each( obj, function( key, value ) {  
6 |   alert( key + ": " + value );  
7 | });
```

Once again, this produces two messages:

```
flammable: inflammable  
duh: no duh
```

We can break the `$.each()` loop at a particular iteration by making the callback function return `false`. Returning *non-false* is the same as a `continue` statement in a for loop; it will skip immediately to the next iteration.

## Examples:

**Example:** *Iterates through the array displaying each number as both a word and numeral*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     div { color:blue; }
6     div#five { color:red; }
7   </style>
8   <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
9 </head>
10 <body>
11
12   <div id="one"></div>
13   <div id="two"></div>
14   <div id="three"></div>
15   <div id="four"></div>
16   <div id="five"></div>
17 <script>
18   var arr = [ "one", "two", "three", "four"
19   var obj = { one:1, two:2, three:3, four:4
20
21   jQuery.each(arr, function() {
22     $("#" + this).text("Mine is " + this +
23     return (this != "three"); // will stop
24   });
25
26   jQuery.each(obj, function(i, val) {
27     $("#" + i).append(document.createTextNode(
28     });
29 </script>
30
```

```
31 | </body>
32 | </html>
```

### Demo

**Example** · Iterates over items in an array, accessing both the current item and its index.

```
1 | $.each( ['a', 'b', 'c'], function(i, l){
2 |     alert( "Index #" + i + ": " + l );
3 | });
```

**Example** · Iterates over the properties in an object, accessing both the current item and its key.

```
1 | $.each( { name: "John", lang: "JS" }, function
2 |     alert( "Key: " + k + ", Value: " + v );
3 | });
```

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# jQuery.error()

Categories: [Internals](#)

`jQuery.error( message )`

*Returns:*

**Description:** *Takes a string and throws an exception containing it.*

`jQuery.error( message )`

version added: [1.4.1](#)

**message**

Type: [String](#)

The message to send out.

This method exists primarily for plugin developers who wish to override it and provide a better display (or more information) for the error messages.

## Example:

*Override jQuery.error for display in Firebug.*

```
1 | jQuery.error = console.error;
```

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# jQuery.extend()

Categories: [Utilities](#)

```
jQuery.extend( target [, object1 ] [, objectN ] )
```

Returns: [Object](#)

**Description:** Merge the contents of two or more objects together into the first object.

```
jQuery.extend( target [, object1 ] [,  
objectN ] )
```

version added: [1.0](#)

**target**

Type: [Object](#)

An object that will receive the new properties if additional objects are passed in or that will extend the jQuery namespace if it is the sole argument.

**object1**

Type: [Object](#)

An object containing additional properties to merge in.

**objectN**

Type: [Object](#)

Additional objects containing properties to merge in.

```
jQuery.extend( [deep ], target,  
object1 [, objectN ] )
```

version added: [1.1.4](#)

**deep**

Type: [Boolean](#)

If true, the merge becomes recursive (aka. deep copy).

**target**

Type: [Object](#)

The object to extend. It will receive the new properties.

**object1**

Type: [Object](#)

An object containing additional properties to merge in.

**objectN**

Type: [Object](#)

## Additional objects containing properties to merge in.

When two or more objects are supplied to `$.extend()`, properties from all of the objects are added to the target object.

If only one argument is supplied to `$.extend()`, this means the target argument was omitted. In this case, the jQuery object itself is assumed to be the target. By doing this, you can add new functions to the jQuery namespace. This can be useful for plugin authors wishing to add new methods to JQuery.

Keep in mind that the target object (first argument) will be modified, and will also be returned from `$.extend()`. If, however, you want to preserve both of the original objects, you can do so by passing an empty object as the target:

```
1 | var object = $.extend({}, object1, object2);
```

The merge performed by `$.extend()` is not recursive by default; if a property of the first object is itself an object or array, it will be completely overwritten by a property with the same key in the second object. The values are not merged. This can be seen in the example below by examining the value of banana. However, by passing `true` for the first function argument, objects will be recursively merged.

**Warning:** Passing `false` for the first argument is not supported.

Undefined properties are not copied. However, properties inherited from the object's prototype *will* be copied over. Properties that are an object constructed via `new MyCustomObject(args)`, or built-in JavaScript types such as Date or RegExp, are not re-constructed and will appear as plain Objects in the resulting object or array.

On a `deep` extend, Object and Array are extended, but object wrappers on primitive types such as String, Boolean, and Number are not.

For needs that fall outside of this behavior, write a custom extend method instead.



## Examples:

**Example:** *Merge two objects, modifying the first.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-1.10.2.min.js"></script>
5 </head>
6 <body>
7
8 <div id="log"></div>
9
10 <script>
11 var object1 = {
12   apple: 0,
13   banana: {weight: 52, price: 100},
14   cherry: 97
15 };
16 var object2 = {
17   banana: {price: 200},
18   durian: 100
19 };
20
21 /* merge object2 into object1 */
22 $.extend(object1, object2);
23
24 var printObj = typeof JSON !== "undefined" ? JSON.stringify : function(obj) {
25   var arr = [];
26   $.each(obj, function(key, val) {
27     var next = key + ": ";
28     next += $.isPlainObject(val) ? printObj(val) : val;
29     arr.push( next );
30   });
31   return "{ " + arr.join(", ") + " }";
```

```
32 };
33
34 $("#log").append( printObj(object1) );
35 </script>
36
37 </body>
38 </html>
```

## Demo

**Example:** Merge two objects recursively, modifying the first.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-
5 </head>
6 <body>
7
8 <div id="log"></div>
9
10 <script>
11 var object1 = {
12   apple: 0,
13   banana: {weight: 52, price: 100},
14   cherry: 97
15 };
16 var object2 = {
17   banana: {price: 200},
18   durian: 100
19 };
20
21 /* merge object2 into object1, recursively */
22 $.extend(true, object1, object2);
23
```

```

24 var printObj = typeof JSON !== "undefined" ? JSON.stringify
25   var arr = [];
26   $.each(obj, function(key, val) {
27     var next = key + ":";
28     next += $.isPlainObject(val) ? printObj(val) : val;
29     arr.push( next );
30   });
31   return "{ " + arr.join(", ") + " }";
32 };
33
34 $("#log").append( printObj(object1) );
35 </script>
36
37 </body>
38 </html>

```

## Demo

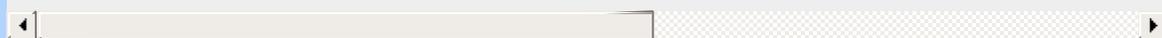
**Example:** *Merge defaults and options, without modifying the defaults. This is a common plugin development pattern.*

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-
5 </head>
6 <body>
7
8 <div id="log"></div>
9
10 <script>
11 var defaults = { validate: false, limit: 5, name: "bar" };
12 var options = { validate: true, name: "bar" };
13
14 /* merge defaults and options, without modify.

```

```
15 var settings = $.extend({}, defaults, options);
16
17 var printObj = typeof JSON !== "undefined" ? JSON.stringify
18   var arr = [];
19   $.each(obj, function(key, val) {
20     var next = key + ": ";
21     next += $.isPlainObject(val) ? printObj(val) : val;
22     arr.push( next );
23   });
24   return "{ " + arr.join(", ") + " }";
25 };
26
27
28 $("#log").append( "<div><b>defaults -- </b>" + printObj(
29 $("#log").append( "<div><b>options -- </b>" + printObj(
30 $("#log").append( "<div><b>settings -- </b>" + printObj(
31
32 </script>
33
34 </body>
35 </html>
```



Demo

POWERED BY HERONOTE

A new version of this book is available!

---

# jQuery.fx.interval

Categories: [Effects](#) > [Custom](#) | [Properties](#) > [Properties of the Global jQuery Object](#)

`jQuery.fx.interval`

Returns: [Number](#)

**Description:** *The rate (in milliseconds) at which animations fire.*

`jQuery.fx.interval`

version added: [1.4.3](#)

This property can be manipulated to adjust the number of frames per second at which animations will run. The default is 13 milliseconds. Making this a lower number could make the animations run smoother in faster browsers (such as Chrome) but there may be performance and CPU implications of doing so.

Since jQuery uses one global interval, no animation should be running or all animations should stop for the change of this property to take effect.

**Note:** `jQuery.fx.interval` currently has no effect in browsers that support the `requestAnimationFrame` property, such as Google Chrome 11. This behavior is subject to change in a future release.

## Example:

*Cause all animations to run with less frames.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     div { width:50px; height:30px; margin:5px; float:left;
6           background:green; }
7   </style>
8   <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
9 </head>
10 <body>
11   <p><input type="button" value="Run"/></p>
12   <div></div>
13   <script>
14     jQuery.fx.interval = 100;
15
16     $("input").click(function(){
17       $("div").toggle( 3000 );
18     });
19   </script>
20
21 </body>
22 </html>
```

Demo

**A new version of this book is available!**

---

# jQuery.fx.off

Categories: [Effects](#) > [Custom](#) | [Properties](#) > [Properties of the Global jQuery Object](#)

---

`jQuery.fx.off`

Returns: [Boolean](#)

**Description:** *Globally disable all animations.*

`jQuery.fx.off`

version added: [1.3](#)

When this property is set to `true`, all animation methods will immediately set elements to their final state when called, rather than displaying an effect. This may be desirable for a couple reasons:

- jQuery is being used on a low-resource device.

- Users are encountering accessibility problems with the animations (see the article [Turn Off Animation](#) for more information).

Animations can be turned back on by setting the property to `false`.

## Example:

### *Toggle animation on and off*

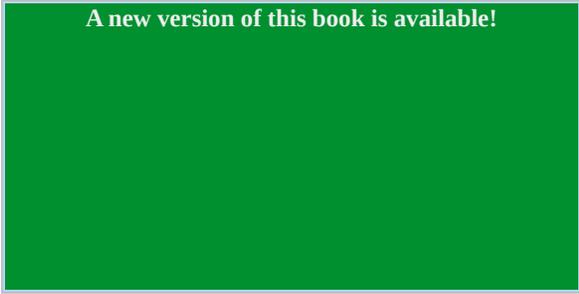
```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5      div { width:50px; height:30px; margin:5px
6            background:green; }
7    </style>
8    <script src="http://code.jquery.com/jquery-
9  </head>
10 <body>
11   <p><input type="button" value="Run"/> <butt
12 <div></div>
13 <script>
14 var toggleFx = function() {
15   $.fx.off = !$.fx.off;
16 };
17 toggleFx();
18
19 $("button").click(toggleFx)
20
21 $("input").click(function(){
22   $("div").toggle("slow");
23 });
24 </script>
25
26 </body>
27 </html>
```

Demo

---

POWERED BY **HERONOTE**

A new version of this book is available!



---

# jQuery.get()

Categories: [Ajax](#) > [Shorthand Methods](#)

`jQuery.get( url [, data ] [, success(data, textStatus, jqXHR) ] [, dataType ] )` Returns: [jqXHR](#)

**Description:** *Load data from the server using a HTTP GET request.*

**jQuery.get( url [, data ] [, success(data, textStatus, jqXHR) ] [, dataType ] )** version added: [1.0](#)

**url**

Type: [String](#)

A string containing the URL to which the request is sent.

**data**

Type: [PlainObject](#) or [String](#)

A plain object or string that is sent to the server with the request.

**success(data, textStatus, jqXHR)**

Type: [Function\(\)](#)

A callback function that is executed if the request succeeds.

**dataType**

Type: [String](#)

The type of data expected from the server. Default: Intelligent Guess (xml, json, script, or html).

This is a shorthand Ajax function, which is equivalent to:

```
1 | $.ajax({
2 |   url: url,
3 |   data: data,
4 |   success: success,
5 |   dataType: dataType
6 | });
```

The `success` callback function is passed the returned data, which will

be an XML root element, text string, JavaScript file, or JSON object, depending on the MIME type of the response. It is also passed the text status of the response.

As of **jQuery 1.5**, the `success` callback function is also passed a ["jqXHR" object](#) (in **jQuery 1.4**, it was passed the `XMLHttpRequest` object). However, since JSONP and cross-domain GET requests do not use XHR, in those cases the `jqXHR` and `textStatus` parameters passed to the success callback are undefined.

Most implementations will specify a success handler:

```
1 | $.get('ajax/test.html', function(data) {  
2 |     $('result').html(data);  
3 |     alert('Load was performed.');
```

This example fetches the requested HTML snippet and inserts it on the page.

## The jqXHR Object

As of **jQuery 1.5**, all of jQuery's Ajax methods return a superset of the `XMLHttpRequest` object. This jQuery XHR object, or "jqXHR," returned by `$.get()` implements the Promise interface, giving it all the properties, methods, and behavior of a Promise (see [Deferred object](#) for more information). The `jqXHR.done()` (for success), `jqXHR.fail()` (for error), and `jqXHR.always()` (for completion, whether success or error) methods take a function argument that is called when the request terminates. For information about the arguments this function receives, see the [jqXHR Object](#) section of the `$.ajax()` documentation.

The Promise interface also allows jQuery's Ajax methods, including `$.get()`, to chain multiple `.done()`, `.fail()`, and `.always()` callbacks on a single request, and even to assign these callbacks after the request may have completed. If the request is already complete, the callback is fired immediately.

```
1 // Assign handlers immediately after making the request
2 // and remember the jqxhr object for this request
3 var jqxhr = $.get("example.php", function() {
4     alert("success");
5 })
6 .done(function() { alert("second success"); })
7 .fail(function() { alert("error"); })
8 .always(function() { alert("finished"); });
9
10 // perform other work here ...
11
12 // Set another completion function for the request above
13 jqxhr.always(function(){ alert("second finished")
```

## Deprecation Notice

The `jqXHR.success()`, `jqXHR.error()`, and `jqXHR.complete()` callback methods introduced in jQuery 1.5 are **deprecated as of jQuery 1.8**. To prepare your code for their eventual removal, use `jqXHR.done()`, `jqXHR.fail()`, and `jqXHR.always()` instead.

## Additional Notes:

Due to browser security restrictions, most "Ajax" requests are subject to the [same origin policy](#); the request can not successfully retrieve data from a different domain, subdomain, or protocol.

If a request with `jQuery.get()` returns an error code, it will fail silently unless the script has also called the global `.ajaxError()` method. Alternatively, as of jQuery 1.5, the `.error()` method of the `jqXHR` object returned by `jQuery.get()` is also available for error handling.

Script and JSONP requests are not subject to the same origin policy restrictions.

## Examples:

**Example:** Request the test.php page, but ignore the return results.

```
1 | $.get("test.php");
```

**Example:** Request the test.php page and send some additional data along (while still ignoring the return results).

```
1 | $.get("test.php", { name: "John", time: "2pm"
```

**Example:** Pass arrays of data to the server (while still ignoring the return results).

```
1 | $.get("test.php", { 'choices[]': ["Jon", "Susan"
```

**Example:** Alert the results from requesting test.php (HTML or XML, depending on what was returned).

```
1 | $.get("test.php", function(data) {  
2 |     alert("Data Loaded: " + data);  
3 | });
```

**Example:** Alert the results from requesting test.php with an additional payload of data (HTML or XML, depending on what was returned).

```
1 $.get("test.cgi", { name: "John", time: "2pm" })
2 .done(function(data) {
3     alert("Data Loaded: " + data);
4 });
```

**Example:** Get the test nbn name contents, which has been returned in json format (`<?nbn echo json encode(array("name"=>"John","time"=>"2pm"));`), and add it to the page.

```
1 $.get("test.php",
2     function(data) {
3         $('body').append("Name: " + data.name );
4         .append("Time: " + data.time );
5     }, "json");
```

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# jQuery.getJSON()

Categories: [Ajax](#) > [Shorthand Methods](#)

`jQuery.getJSON( url [, data ] [, success(data, textStatus, jqXHR) ] )` Returns: `jqXHR`

**Description:** Load JSON-encoded data from the server using a GET HTTP request.

**jQuery.getJSON( url [, data ] [, success(data, textStatus, jqXHR) ] )** version added: **1.0**

**url**

Type: [String](#)

A string containing the URL to which the request is sent.

**data**

Type: [PlainObject](#)

A plain object or string that is sent to the server with the request.

**success(data, textStatus, jqXHR)**

Type: [Function\(\)](#)

A callback function that is executed if the request succeeds.

This is a shorthand Ajax function, which is equivalent to:

```
1 | $.ajax({
2 |   dataType: "json",
3 |   url: url,
4 |   data: data,
5 |   success: success
6 | });
```

Data that is sent to the server is appended to the URL as a query string. If the value of the `data` parameter is a plain object, it is converted to a string and url-encoded before it is appended to the URL.

Most implementations will specify a success handler:

```
1 $.getJSON('ajax/test.json', function(data) {
2   var items = [];
3
4   $.each(data, function(key, val) {
5     items.push('<li id="' + key + '">' + val
6   });
7
8   $('<ul/>', {
9     'class': 'my-new-list',
10    html: items.join('')
11  }).appendTo('body');
12 });
```

This example, of course, relies on the structure of the JSON file:

```
1 {
2   "one": "Singular sensation",
3   "two": "Beady little eyes",
4   "three": "Little birds pitch by my doorstep"
5 }
```

Using this structure, the example loops through the requested data, builds an unordered list, and appends it to the body.

The `success` callback is passed the returned data, which is typically a JavaScript object or array as defined by the JSON structure and parsed using the `$.parseJSON()` method. It is also passed the text status of the response.

**As of jQuery 1.5**, the `success` callback function receives a ["jqXHR" object](#) (in **jQuery 1.4**, it received the `XMLHttpRequest` object). However, since JSONP and cross-domain GET requests do not use XHR, in those cases the `jqXHR` and `textStatus` parameters passed to the success callback are undefined.

**Important:** As of jQuery 1.4, if the JSON file contains a syntax error, the request will usually fail silently. Avoid frequent hand-editing of JSON data for this reason. JSON is a data-interchange format with syntax rules that are stricter than those of JavaScript's object literal notation. For example, all strings represented in JSON, whether they are properties or values, must be enclosed in double-quotes. For details on the JSON format, see <http://json.org/>.

## JSONP

If the URL includes the string "callback=?" (or similar, as defined by the server-side API), the request is treated as JSONP instead. See the discussion of the `jsonp` data type in `$.ajax()` for more details.

## The jqXHR Object

As of jQuery 1.5, all of jQuery's Ajax methods return a superset of the `XMLHttpRequest` object. This jQuery XHR object, or "jqXHR," returned by `$.getJSON()` implements the Promise interface, giving it all the properties, methods, and behavior of a Promise (see [Deferred object](#) for more information). For convenience and consistency with the callback names used by `$.ajax()`, it provides `.error()`, `.success()`, and `.complete()` methods. These methods take a function argument that is called when the request terminates, and the function receives the same arguments as the correspondingly-named `$.ajax()` callback.

The Promise interface in jQuery 1.5 also allows jQuery's Ajax methods, including `$.getJSON()`, to chain multiple `.success()`, `.complete()`, and `.error()` callbacks on a single request, and even

to assign these callbacks after the request may have completed. If the request is already complete, the callback is fired immediately.

```
1 // Assign handlers immediately after making a request
2 // and remember the jqxhr object for this request
3 var jqxhr = $.getJSON("example.json", function() {
4     alert("success");
5 });
6 .success(function() { alert("second success"); })
7 .error(function() { alert("error"); })
8 .complete(function() { alert("complete"); });
9
10 // perform other work here ...
11
12 // Set another completion function for the request
13 jqxhr.complete(function(){ alert("second complete"); });
```

## Additional Notes:

Due to browser security restrictions, most "Ajax" requests are subject to the [same origin policy](#); the request can not successfully retrieve data from a different domain, subdomain, or protocol.

Script and JSONP requests are not subject to the same origin policy restrictions.

## Examples:

**Example:** Loads the four most recent cat pictures from the Flickr JSONP API.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>img{ height: 100px; float: left; }</style>
5   <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
6 </head>
7 <body>
8   <div id="images">
9
10  </div>
11 <script>
12 $.getJSON("http://api.flickr.com/services/feeds/photos_public.gne?tags=cat&tagmode=all&format=json&callback=?",
13 {
14   tags: "mount rainier",
15   tagmode: "any",
16   format: "json"
17 },
18 function(data) {
19   $.each(data.items, function(i,item){
20     $("<img/>").attr("src", item.media.m).appendTo("#images");
21     if ( i == 3 ) return false;
22   });
23 });</script>
24
25 </body>
26 </html>
```

Demo

**Example:** I load the JSON data from test.js and access a name from the returned JSON data.

```
1 | $.getJSON("test.js", function(json) {  
2 |     alert("JSON Data: " + json.users[3].name);  
3 | });
```

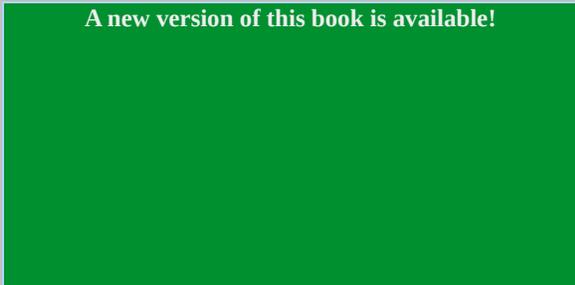
**Example:** I load the JSON data from test.js, passing along additional data and access a name from the returned JSON data.

```
1 | $.getJSON("test.js", { name: "John", time: "2p  
2 |     alert("JSON Data: " + json.users[3].name);  
3 | });
```



POWERED BY HERONOTE

A new version of this book is available!



---

# jQuery.getScript()

Categories: [Ajax](#) > [Shorthand Methods](#)

```
jQuery.getScript( url [, success(script, textStatus, jqXHR) ] )  
Returns: jqXHR
```

**Description:** Load a JavaScript file from the server using a GET HTTP request, then execute it.

```
jQuery.getScript( url [, version added: 1.0  
success(script, textStatus, jqXHR) ] )
```

**url**

Type: [String](#)

A string containing the URL to which the request is sent.

**success(script, textStatus, jqXHR)**

Type: [Function\(\)](#)

A callback function that is executed if the request succeeds.

This is a shorthand Ajax function, which is equivalent to:

```
1 | $.ajax({  
2 |   url: url,  
3 |   dataType: "script",  
4 |   success: success  
5 | });
```

The script is executed in the global context, so it can refer to other variables and use jQuery functions. Included scripts can have some impact on the current page.

### Success Callback

The callback is fired once the script has been loaded but not necessarily executed.

```
1 | $(".result").html("<p>Lorem ipsum dolor sit ar
```

Scripts are included and run by referencing the file name:

```
1 | $.getScript("ajax/test.js", function(data, text) {
2 |     console.log(data); //data returned
3 |     console.log(textStatus); //success
4 |     console.log(jqxhr.status); //200
5 |     console.log('Load was performed.');
```

## Handling Errors

As of jQuery 1.5, you may use `.fail()` to account for errors:

```
1 | $.getScript("ajax/test.js")
2 | .done(function(script, textStatus) {
3 |     console.log( textStatus );
4 | })
5 | .fail(function(jqxhr, settings, exception) {
6 |     $( "div.log" ).text( "Triggered ajaxError handler");
7 | });
```

Prior to jQuery 1.5, the global `.ajaxError()` callback event had to be used in order to handle `$.getScript()` errors:

```
1 | $( "div.log" ).ajaxError(function(e, jqxhr, settings) {
2 |     if (settings.dataType=='script') {
3 |         $(this).text( "Triggered ajaxError handler");
4 |     }
5 | });
```

## Caching Responses

By default, `$.getScript()` sets the cache setting to `false`. This appends a timestamped query parameter to the request URL to ensure that the browser downloads the script each time it is requested. You can override this feature by setting the cache property globally using `$.ajaxSetup()`:

```
1 | $.ajaxSetup({  
2 |   cache: true  
3 | });
```

Alternatively, you could define a new method that uses the more flexible `$.ajax()` method.

## Examples:

**Example:** Define a `$.cachedScript()` method that allows fetching a cached script:

```
1 | jQuery.cachedScript = function(url, options)
2 |
3 |     // allow user to set any option except for
4 |     options = $.extend(options || {}, {
5 |         dataType: "script",
6 |         cache: true,
7 |         url: url
8 |     });
9 |
10 |     // Use $.ajax() since it is more flexible to
11 |     // Return the jqXHR object so we can chain
12 |     return jQuery.ajax(options);
13 | };
14 |
15 | // Usage
16 | $.cachedScript("ajax/test.js").done(function()
17 |     console.log( textStatus );
18 | });
```

**Example:** Load the [official jQuery Color Animation plugin](#) dynamically and bind some color animations to occur once the new functionality is loaded.

```
1 | <!DOCTYPE html>
2 | <html>
3 | <head>
4 |     <style>
5 |     .block {
```

```
6     background-color: blue;
7     width: 150px;
8     height: 70px;
9     margin: 10px;
10  }</style>
11  <script src="http://code.jquery.com/jquery-latest.js"
12  </head>
13  <body>
14
15  <button id="go">&raquo; Run</button>
16
17  <div class="block"></div>
18
19  <script>
20  (function() {
21    var url = "https://raw.github.com/jquery/jquery-color.
22    $.getScript(url, function() {
23      $("#go").click(function(){
24        $(".block")
25          .animate( { backgroundColor: "rgb(255, 180, 180)
26          .delay(500)
27          .animate( { backgroundColor: "olive" },
28          .delay(500)
29          .animate( { backgroundColor: "#00f" },
30        });
31      });
32    })();
33  </script>
34
35  </body>
36  </html>
```

Demo

**A new version of this book is available!**

---

# jQuery.globalEval()

Categories: [Utilities](#)

`jQuery.globalEval( code )`

*Returns:*

**Description:** *Execute some JavaScript code globally.*

`jQuery.globalEval( code )`

version added: [1.0.4](#)

**code**

Type: [String](#)

The JavaScript code to execute.

This method behaves differently from using a normal JavaScript `eval()` in that it's executed within the global context (which is important for loading external scripts dynamically).

## Example:

*Execute a script in the global context.*

```
1 | function test(){  
2 |     jQuery.globalEval("var newVar = true;")  
3 | }  
4 | test();  
5 | // newVar === true
```

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# jQuery.grep()

Categories: [Utilities](#)

`jQuery.grep( array, function(elementOfArray, indexInArray) [, invert ] )` Returns: [Array](#)

**Description:** *Finds the elements of an array which satisfy a filter function. The original array is not affected.*

**jQuery.grep( array, function(elementOfArray, indexInArray) [, invert ] )** version added: [1.0](#)

**array**

Type: [Array](#)

The array to search through.

**function(elementOfArray, indexInArray)**

Type: [Function\(\)](#)

The function to process each item against. The first argument to the function is the item, and the second argument is the index. The function should return a Boolean value. `this` will be the global window object.

**invert**

Type: [Boolean](#)

If "invert" is false, or not provided, then the function returns an array consisting of all elements for which "callback" returns true. If "invert" is true, then the function returns an array consisting of all elements for which "callback" returns false.

The `$.grep()` method removes items from an array as necessary so that all remaining items pass a provided test. The test is a function that is passed an array item and the index of the item within the array. Only if the test returns true will the item be in the result array.

The filter function will be passed two arguments: the current array item and its index. The filter function must return 'true' to include the item in the result array.

## Examples:

**Example:** *Filters the original array of numbers leaving that are not 5 and have an index greater than 4. Then it removes all 9s.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <style>
5   div { color:blue; }
6   p { color:green; margin:0; }
7   span { color:red; }
8 </style>
9 <script src="http://code.jquery.com/jquery-
10 </head>
11 <body>
12   <div></div>
13   <p></p>
14   <span></span>
15
16 <script>
17 var arr = [ 1, 9, 3, 8, 6, 1, 5, 9, 4, 7, 3, 5 ];
18 $("div").text(arr.join(", "));
19
20 arr = jQuery.grep(arr, function(n, i){
21   return (n != 5 && i > 4);
22 });
23 $("p").text(arr.join(", "));
24
25 arr = jQuery.grep(arr, function (a) { return
26 $("span").text(arr.join(", "));
27
28 </script>
29
```

```
30 | </body>
31 | </html>
```

### Demo

**Example** • *Filter an array of numbers to include only numbers bigger than zero.*

```
1 | $.grep( [0, 1, 2], function(n, i){
2 |     return n > 0;
3 | });
```

### Result:

```
1 | [1, 2]
```

**Example** • *Filter an array of numbers to include numbers that are not bigger than zero.*

```
1 | $.grep( [0, 1, 2], function(n, i){
2 |     return n > 0;
3 | }, true);
```

### Result:

```
1 | [0]
```

**A new version of this book is available!**

---

# jQuery.hasData()

Categories: [Data](#)

`jQuery.hasData( element )`

Returns: [Boolean](#)

**Description:** *Determine whether an element has any jQuery data associated with it.*

**jQuery.hasData( element )**

version added: [1.5](#)

**element**

Type: [Element](#)

A DOM element to be checked for data.

The `jQuery.hasData()` method provides a way to determine if an element currently has any values that were set using `jQuery.data()`. If no data is associated with an element (there is no data object at all or the data object is empty), the method returns `false`; otherwise it returns `true`.

The primary advantage of `jQuery.hasData(element)` is that it does not create and associate a data object with the element if none currently exists. In contrast, `jQuery.data(element)` always returns a data object to the caller, creating one if no data object previously existed.

Note that jQuery's event system uses the jQuery data API to store event handlers. Therefore, binding an event to an element using `.on()`, `.bind()`, `.live()`, `.delegate()`, or one of the shorthand event methods also associates a data object with that element.

## Example:

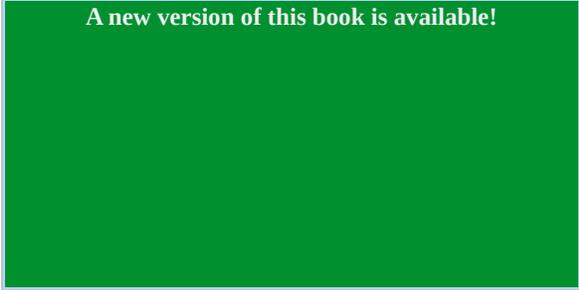
*Set data on an element and see the results of `hasData`.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-1.10.2.min.js"></script>
5 </head>
6 <body>
7   <p>Results: </p>
8   <script>
9     var $p = jQuery("p"), p = $p[0];
10    $p.append(jQuery.hasData(p)+" "); /* false */
11
12    $.data(p, "testing", 123);
13    $p.append(jQuery.hasData(p)+" "); /* true */
14
15    $.removeData(p, "testing");
16    $p.append(jQuery.hasData(p)+" "); /* false */
17
18    $p.on('click', function() {});
19    $p.append(jQuery.hasData(p)+" "); /* true */
20
21    $p.off('click');
22    $p.append(jQuery.hasData(p)+" "); /* false */
23
24  </script>
25
26 </body>
27 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

# jQuery.holdReady()

Categories: [Core](#)

`jQuery.holdReady( hold )`

Returns: `undefined`

**Description:** *Holds or releases the execution of jQuery's ready event.*

**jQuery.holdReady( hold )**

version added: [1.6](#)

**hold**

Type: [Boolean](#)

Indicates whether the ready hold is being requested or released

The `$.holdReady()` method allows the caller to delay jQuery's ready event. This *advanced feature* would typically be used by dynamic script loaders that want to load additional JavaScript such as jQuery plugins before allowing the ready event to occur, even though the DOM may be ready. This method must be called early in the document, such as in the `<head>` immediately after the jQuery script tag. Calling this method after the ready event has already fired will have no effect.

To delay the ready event, first call `$.holdReady(true)`. When the ready event should be released to execute, call `$.holdReady(false)`. Note that multiple holds can be put on the ready event, one for each `$.holdReady(true)` call. The ready event will not actually fire until all holds have been released with a corresponding number of `$.holdReady(false)` calls *and* the normal document ready conditions are met. (See [ready](#) for more information.)

## Example:

*Delay the ready event until a custom plugin has loaded.*

```
1 | $.holdReady(true);  
2 | $.getScript("myplugin.js", function() {  
3 |     $.holdReady(false);  
4 | });
```

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# jQuery.inArray()

Categories: [Utilities](#)

`jQuery.inArray( value, array [, fromIndex ] )`

Returns: [Number](#)

**Description:** Search for a specified value within an array and return its index (or -1 if not found).

**jQuery.inArray( value, array [, fromIndex ] )**

version added: [1.2](#)

**value**

Type: [Anything](#)

The value to search for.

**array**

Type: [Array](#)

An array through which to search.

**fromIndex**

Type: [Number](#)

The index of the array at which to begin the search. The default is 0, which will search the whole array.

The `$.inArray()` method is similar to JavaScript's native `.indexOf()` method in that it returns -1 when it doesn't find a match. If the first element within the array matches `value`, `$.inArray()` returns 0.

Because JavaScript treats 0 as loosely equal to false (i.e. `0 == false`, but `0 !== false`), if we're checking for the presence of `value` within `array`, we need to check if it's not equal to (or greater than) -1.

## Example:

*Report the index of some elements in the array.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     div { color:blue; }
6     span { color:red; }
7   </style>
8   <script src="http://code.jquery.com/jquery-
9 </head>
10 <body>
11
12 <div>"John" found at <span></span></div>
13 <div>4 found at <span></span></div>
14 <div>"Karl" not found, so <span></span></div>
15 <div>"Pete" is in the array, but not at or af
16 <script>var arr = [ 4, "Pete", 8, "John" ];
17 var $spans = $("span");
18 $spans.eq(0).text(jQuery.inArray("John", arr)
19 $spans.eq(1).text(jQuery.inArray(4, arr));
20 $spans.eq(2).text(jQuery.inArray("Karl", arr)
21 $spans.eq(3).text(jQuery.inArray("Pete", arr,
22 </script>
23
24 </body>
25 </html>
```

Demo

**A new version of this book is available!**

---

# jQuery.isArray()

Categories: [Utilities](#)

`jQuery.isArray( obj )`

Returns: [boolean](#)

**Description:** *Determine whether the argument is an array.*

`jQuery.isArray( obj )`

version added: [1.3](#)

**obj**

Type: [Object](#)

Object to test whether or not it is an array.

`$.isArray()` returns a Boolean indicating whether the object is a JavaScript array (not an array-like object, such as a jQuery object).

## Example:

*Finds out if the parameter is an array.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
5 </head>
6 <body>
7   Is [] an Array? <b></b>
8   <script>$("b").append( "" + $.isArray([]) );</script>
9
10 </body>
11 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# jQuery.isEmptyObject()

Categories: [Utilities](#)

`jQuery.isEmptyObject( object )`

Returns: [Boolean](#)

**Description:** *Check to see if an object is empty (contains no enumerable properties).*

`jQuery.isEmptyObject( object )`

version added: [1.4](#)

**object**

Type: [Object](#)

The object that will be checked to see if it's empty.

As of jQuery 1.4 this method checks both properties on the object itself and properties inherited from prototypes (in that it doesn't use `hasOwnProperty`). The argument should always be a plain JavaScript `object` as other types of object (DOM elements, primitive strings/numbers, host objects) may not give consistent results across browsers. To determine if an object is a plain JavaScript object, use `$.isPlainObject()`

## Example:

*Check an object to see if it's empty.*

```
1 | jQuery.isEmptyObject({}) // true
2 |     jQuery.isEmptyObject({ foo: "bar" }) //
```

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# jQuery.isFunction()

Categories: [Utilities](#)

`jQuery.isFunction( obj )`

Returns: [boolean](#)

**Description:** *Determine if the argument passed is a Javascript function object.*

`jQuery.isFunction( obj )`

version added: [1.2](#)

**obj**

Type: [PlainObject](#)

Object to test whether or not it is a function.

**Note:** As of jQuery 1.3, functions provided by the browser like `alert()` and DOM element methods like `getAttribute()` are not guaranteed to be detected as functions in browsers such as Internet Explorer.

## Examples:

**Example:** *Test a few parameter examples.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5    div { color:blue; margin:2px; font-size:14px; }
6    span { color:red; }
7    </style>
8    <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
9  </head>
10 <body>
11
12   <div>jQuery.isFunction(objs[0]) = <span></span>
13
14   <div>jQuery.isFunction(objs[1]) = <span></span>
15   <div>jQuery.isFunction(objs[2]) = <span></span>
16   <div>jQuery.isFunction(objs[3]) = <span></span>
17
18   <div>jQuery.isFunction(objs[4]) = <span></span>
19
20   <script>
21   function stub() {
22   }
23   var objs = [
24     function () {},
25     { x:15, y:20 },
26     null,
27     stub,
28     "function"
29   ];
30
31   jQuery.each(objs, function (i) {
```

```
32 |   var isFunc = jQuery.isFunction(objs[i]);
33 |   $("span").eq(i).text(isFunc);
34 |   });
35 | </script>
36 |
37 | </body>
38 | </html>
```

Demo

**Example:** *Finds out if the parameter is a function.*

```
1 | $.isFunction(function({}));
```

**Result:**

```
1 | true
```

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# jQuery.isNumeric()

Categories: [Utilities](#)

`jQuery.isNumeric( value )`

Returns: [Boolean](#)

**Description:** *Determines whether its argument is a number.*

`jQuery.isNumeric( value )`

version added: [1.7](#)

**value**

Type: [PlainObject](#)

The value to be tested.

The `$.isNumeric()` method checks whether its argument represents a numeric value. If so, it returns `true`. Otherwise it returns `false`. The argument can be of any type.

## Example:

*Sample return values of \$.isNumeric with various inputs.*

```
1 $.isNumeric("-10"); // true
2 $.isNumeric(16); // true
3 $.isNumeric(0xFF); // true
4 $.isNumeric("0xFF"); // true
5 $.isNumeric("8e5"); // true (exponential notation)
6 $.isNumeric(3.1415); // true
7 $.isNumeric(+10); // true
8 $.isNumeric(0144); // true (octal integer literal)
9 $.isNumeric(""); // false
10 $.isNumeric({}); // false (empty object)
11 $.isNumeric(NaN); // false
12 $.isNumeric(null); // false
13 $.isNumeric(true); // false
14 $.isNumeric(Infinity); // false
15 $.isNumeric(undefined); // false
```

POWERED BY HERONOTE

A new version of this book is available!

---

# jQuery.isPlainObject()

Categories: [Utilities](#)

`jQuery.isPlainObject( object )`

Returns: [Boolean](#)

**Description:** Check to see if an object is a plain object (created using `"{}"` or `"new Object"`).

`jQuery.isPlainObject( object )`

version added: [1.4](#)

**object**

Type: [PlainObject](#)

The object that will be checked to see if it's a plain object.

**Note:** Host objects (or objects used by browser host environments to complete the execution environment of ECMAScript) have a number of inconsistencies which are difficult to robustly feature detect cross-platform. As a result of this, `$.isPlainObject()` may evaluate inconsistently across browsers in certain instances.

An example of this is a test against `document.location` using `$.isPlainObject()` as follows:

```
1 | console.log($.isPlainObject(document.location))
```

which throws an invalid pointer exception in IE8. With this in mind, it's important to be aware of any of the gotchas involved in using `$.isPlainObject()` against older browsers. A couple basic examples that do function correctly cross-browser can be found below.

## Example:

*Check an object to see if it's a plain object.*

```
1 | jQuery.isPlainObject({}) // true
2 | jQuery.isPlainObject("test") // false
```

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# jQuery.isWindow()

Categories: [Utilities](#)

`jQuery.isWindow( obj )`

Returns: [boolean](#)

**Description:** *Determine whether the argument is a window.*

`jQuery.isWindow( obj )`

version added: [1.4.3](#)

**obj**

Type: [PlainObject](#)

Object to test whether or not it is a window.

This is used in a number of places in jQuery to determine if we're operating against a browser window (such as the current window or an iframe).

## Example:

*Finds out if the parameter is a window.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <script src="http://code.jquery.com/jquery-
5  </head>
6  <body>
7    Is 'window' a window? <b></b>
8    <script>$("b").append( "" + $.isWindow(window
9
10 </body>
11 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# jQuery.isXMLDoc()

Categories: [Utilities](#)

`jQuery.isXMLDoc( node )`

Returns: [Boolean](#)

**Description:** *Check to see if a DOM node is within an XML document (or is an XML document).*

**jQuery.isXMLDoc( node )**

version added: [1.1.4](#)

**node**

Type: [Element](#)

The DOM node that will be checked to see if it's in an XML document.

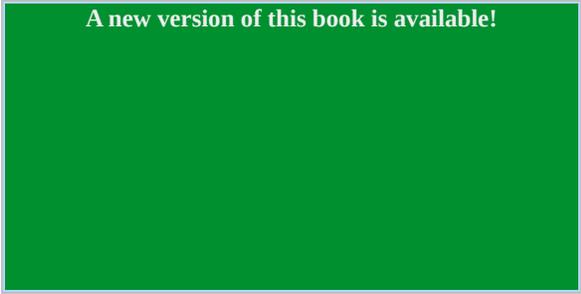
## Example:

*Check an object to see if it's in an XML document.*

```
1 | jQuery.isXMLDoc(document) // false
2 |     jQuery.isXMLDoc(document.body) // false
```

POWERED BY **HERONOTE**

A new version of this book is available!



---

# jQuery.makeArray()

Categories: [Utilities](#)

`jQuery.makeArray( obj )`

Returns: [Array](#)

**Description:** *Convert an array-like object into a true JavaScript array.*

`jQuery.makeArray( obj )`

version added: [1.2](#)

**obj**

Type: [PlainObject](#)

Any object to turn into a native Array.

Many methods, both in jQuery and in JavaScript in general, return objects that are array-like. For example, the jQuery factory function `$()` returns a jQuery object that has many of the properties of an array (a length, the `[]` array access operator, etc.), but is not exactly the same as an array and lacks some of an array's built-in methods (such as `.pop()` and `.reverse()`).

Note that after the conversion, any special features the object had (such as the jQuery methods in our example) will no longer be present. The object is now a plain array.

## Examples:

**Example:** Turn a collection of HTML Elements into an Array of them.

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5      div { color:red; }
6    </style>
7    <script src="http://code.jquery.com/jquery-...
8  </head>
9  <body>
10   <div>First</div>
11   <div>Second</div>
12   <div>Third</div>
13
14   <div>Fourth</div>
15  <script>
16    var elems = document.getElementsByTagName
17    var arr = jQuery.makeArray(elems);
18    arr.reverse(); // use an Array method on .
19    $(arr).appendTo(document.body);
20  </script>
21
22  </body>
23  </html>
```

Demo

**Example:** Turn a jQuery object into an array

```
1  var obj = $('li');
```

```
2 |   var arr = $.makeArray(obj);
```

**Result:**

```
1 | (typeof obj === 'object' && obj.jquery) === true  
2 | jQuery.isArray(arr) === true;
```

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# jQuery.map()

Categories: [Utilities](#)

```
jQuery.map( array, callback(elementOfArray, indexInArray) )  
Returns: Array
```

**Description:** *Translate all items in an array or object to new array of items.*

```
jQuery.map( array, callback(elementOfArray, indexInArray) )
```

version added: [1.0](#)

**array**

Type: [Array](#)

The Array to translate.

**callback(elementOfArray, indexInArray)**

Type: [Function\(\)](#)

The function to process each item against. The first argument to the function is the array item, the second argument is the index in array. The function can return any value. Within the function, `this` refers to the global (window) object.

```
jQuery.map( arrayOrObject, callback( value, indexOrKey ) )
```

version added: [1.6](#)

**arrayOrObject**

Type: [Array](#) or [Object](#)

The Array or Object to translate.

**callback( value, indexOrKey )**

Type: [Function\(\)](#)

The function to process each item against. The first argument to the function is the value; the second argument is the index or key of the array or object property. The function can return any value to add to the array. A returned array will be flattened into the resulting array. Within the function, `this` refers to the global (window) object.

If you wish to process a jQuery object — for example, `$('#div').map( callback );` — use [.map\(\)](#) instead.

The `$.map()` method applies a function to each item in an array or

object and maps the results into a new array. Prior to jQuery 1.6, `$.map()` supports traversing *arrays only*. As of jQuery 1.6 it also traverses objects.

Array-like objects — those with a `.length` property *and* a value on the `.length - 1` index — must be converted to actual arrays before being passed to `$.map()`. The jQuery library provides [\\$.makeArray\(\)](#) for such conversions.

```
1 // The following object masquerades as an array
2 var fakeArray = { "length": 1, 0: "Addy", 1: '
3
4 // Therefore, convert it to a real array
5 var realArray = $.makeArray( fakeArray )
6
7 // Now it can be used reliably with $.map()
8 $.map( realArray, function(val, i) {
9     // do something
10 });
```

The translation function that is provided to this method is called for each top-level element in the array or object and is passed two arguments: The element's value and its index or key within the array or object.

The function can return:

- the translated value, which will be mapped to the resulting array
- `null` or `undefined`, to remove the item
- an array of values, which will be flattened into the full array

## Examples:

**Example:** Use `$.map()` to change the values of an array.

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5    div { color:blue; }
6    p { color:green; margin:0; }
7    span { color:red; }
8    </style>
9    <script src="http://code.jquery.com/jquery-
10 </head>
11 <body>
12   <div></div>
13   <p></p>
14   <span></span>
15
16 <script>
17   var arr = [ "a", "b", "c", "d", "e" ];
18   $("div").text(arr.join(", "));
19
20   arr = jQuery.map(arr, function(n, i){
21     return (n.toUpperCase() + i);
22   });
23   $("p").text(arr.join(", "));
24
25   arr = jQuery.map(arr, function (a) {
26     return a + a;
27   });
28   $("span").text(arr.join(", "));
29
30 </script>
31
```

```
32 | </body>
33 | </html>
```

### Demo

**Example:** *Map the original array to a new one and add 4 to each value.*

```
1 | $.map( [0, 1, 2], function(n){
2 |     return n + 4;
3 | });
```

### Result:

```
1 | [4, 5, 6]
```

**Example:** *Map the original array to a new one, adding 1 to each value if it is bigger than zero and removing it if not.*

```
1 | $.map( [0, 1, 2], function(n){
2 |     return n > 0 ? n + 1 : null;
3 | });
```

### Result:

```
1 | [2, 3]
```

**Example:** *Map the original array to a new one: each element is added with its original value and the value plus one.*

```
1 | $.map( [0, 1, 2], function(n){
2 |     return [ n, n + 1 ];
3 | });
```

**Result:**

```
1 | [0, 1, 1, 2, 2, 3]
```

**Example:** *Map the original object to a new array and double each value.*

```
1 | var dimensions = { width: 10, height: 15, length: 20 };
2 | dimensions = $.map( dimensions, function( value, key ){
3 |     return value * 2;
4 | });
```

**Result:**

```
1 | [20, 30, 40]
```

**Example:** *Map an object's keys to an array.*

```
1 | var dimensions = { width: 10, height: 15, length: 20 };
2 | var keys = $.map( dimensions, function( value, key ){
3 |     return key;
4 | });
```

**Result:**

```
1 |
```

```
1 | ["width", "height", "length"]
```

**Example:** Map the original array to a new one; each element is squared.

```
1 | $.map( [0, 1, 2, 3], function (a) {  
2 |     return a * a;  
3 | });
```

**Result:**

```
1 | [0, 1, 4, 9]
```

**Example:** Map the original array to a new one, removing numbers less than 50 by returning `null` and subtracting 45 from the rest.

```
1 | $.map( [0, 1, 52, 97], function (a) {  
2 |     return (a > 50 ? a - 45 : null);  
3 | });
```

**Result:**

```
1 | [7, 52]
```

**Example:** Augment the resulting array by returning an array inside the function.

```
1 | var array = [0, 1, 52, 97];  
2 | array = $.map(array, function(a, index) {  
3 |     return [a - 45, index];
```

```
4 | });
```

**Result:**

```
1 | [-45, 0, -44, 1, 7, 2, 52, 3]
```

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# jQuery.merge()

Categories: [Utilities](#)

`jQuery.merge( first, second )`

Returns: [Array](#)

**Description:** Merge the contents of two arrays together into the first array.

`jQuery.merge( first, second )`

version added: [1.0](#)

**first**

Type: [Array](#)

The first array to merge, the elements of second added.

**second**

Type: [Array](#)

The second array to merge into the first, unaltered.

The `$.merge()` operation forms an array that contains all elements from the two arrays. The orders of items in the arrays are preserved, with items from the second array appended. The `$.merge()` function is destructive. It alters the first parameter to add the items from the second.

If you need the original first array, make a copy of it before calling `$.merge()`. Fortunately, `$.merge()` itself can be used for this duplication:

```
1 | var newArray = $.merge([], oldArray);
```

This shortcut creates a new, empty array and merges the contents of `oldArray` into it, effectively cloning the array.

Prior to jQuery 1.4, the arguments should be true Javascript Array objects; use `$.makeArray` if they are not.

## Examples:

**Example:** *Merges two arrays, altering the first argument.*

```
1 | $.merge( [0,1,2], [2,3,4] )
```

**Result:**

```
1 | [0,1,2,2,3,4]
```

**Example:** *Merges two arrays, altering the first argument.*

```
1 | $.merge( [3,2,1], [4,3,2] )
```

**Result:**

```
1 | [3,2,1,4,3,2]
```

**Example:** *Merges two arrays, but uses a copy, so the original isn't altered.*

```
1 | var first = ['a', 'b', 'c'];  
2 | var second = ['d', 'e', 'f'];  
3 | $.merge( $.merge([], first), second);
```

**Result:**

```
1 | ["a", "b", "c", "d", "e", "f"]
```

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# jQuery.noConflict()

Categories: [Core](#) | [Miscellaneous](#) > [Setup Methods](#)

jQuery.noConflict( [removeAll ] )

Returns: [Object](#)

**Description:** Relinquish jQuery's control of the `$` variable.

jQuery.noConflict( [removeAll ] )

version added: [1.0](#)

#### removeAll

Type: [Boolean](#)

A Boolean indicating whether to remove all jQuery variables from the global scope (including jQuery itself).

Many JavaScript libraries use `$` as a function or variable name, just as jQuery does. In jQuery's case, `$` is just an alias for `jQuery`, so all functionality is available without using `$`. If you need to use another JavaScript library alongside jQuery, return control of `$` back to the other library with a call to `$.noConflict()`. Old references of `$` are saved during jQuery initialization; `noConflict()` simply restores them.

If for some reason two versions of jQuery are loaded (which is not recommended), calling `$.noConflict(true)` from the second version will return the globally scoped jQuery variables to those of the first version.

```
1 | <script type="text/javascript" src="other_lib
2 | <script type="text/javascript" src="jquery.js"
3 | <script type="text/javascript">
4 |     $.noConflict();
5 |     // Code that uses other library's $ can fol
6 | </script>
```

This technique is especially effective in conjunction with the `.ready()` method's ability to alias the jQuery object, as within callback passed to `.ready()` you can use `$` if you wish without fear of conflicts later:

```
1 <script type="text/javascript" src="other_lib
2 <script type="text/javascript" src="jquery.js"
3 <script type="text/javascript">
4     $.noConflict();
5     jQuery(document).ready(function($) {
6         // Code that uses jQuery's $ can follow here
7     });
8     // Code that uses other library's $ can follow here
9 </script>
```

If necessary, you can free up the `jQuery` name as well by passing `true` as an argument to the method. This is rarely necessary, and if you must do this (for example, if you need to use multiple versions of the jQuery library on the same page), you need to consider that most plug-ins rely on the presence of the `jQuery` variable and may not operate correctly in this situation.

## Examples:

**Example:** *Man the original object that was referenced by \$ back to \$.*

```
1 | jQuery.noConflict();
2 | // Do something with jQuery
3 | jQuery("div p").hide();
4 | // Do something with another library's $()
5 | $("content").style.display = 'none';
```

**Example:** *Revert the \$ alias and then create and execute a function to provide the \$ as a jQuery alias inside the function's scope. Inside the function the original \$ object is not available. This works well for most plugins that don't rely on any other library.*

```
1 | jQuery.noConflict();
2 | (function($) {
3 |     $(function() {
4 |         // more code using $ as alias to jQuery
5 |     });
6 | })(jQuery);
7 | // other code using $ as an alias to the other
```

**Example:** *Create a different alias instead of jQuery to use in the rest of the script.*

```
1 | var j = jQuery.noConflict();
2 | // Do something with jQuery
3 | j("div p").hide();
4 | // Do something with another library's $()
```

```
5 | $("content").style.display = 'none';
```

**Example:** *Completely move jQuery to a new namespace in another object.*

```
1 | var dom = {};  
2 | dom.query = jQuery.noConflict(true);
```

**Result:**

```
1 | // Do something with the new jQuery  
2 | dom.query("div p").hide();  
3 | // Do something with another library's $()  
4 | $("content").style.display = 'none';  
5 | // Do something with another version of jQuery  
6 | jQuery("div > p").hide();
```

**Example:** *I load two versions of jQuery (not recommended) Then restore jQuery's globally scoped variables to the first loaded jQuery.*

```
1 | <!DOCTYPE html>  
2 | <html>  
3 | <head>  
4 |   <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>  
5 | </head>  
6 | <body>  
7 |  
8 | <div id="log">  
9 |   <h3>Before $.noConflict(true)</h3>  
10 | </div>  
11 | <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>  
12 |
```

```
13 <script>
14 var $log = $( "#log" );
15
16 $log.append( "2nd loaded jQuery version ($):" + $.fn.j
17
18 /*
19 Restore globally scoped jQuery variables to the first v
20 (the newer version)
21 */
22 jq162 = jQuery.noConflict(true);
23
24 $log.append( "<h3>After $.noConflict(true)</h3>"
25 $log.append( "1st loaded jQuery version ($):" + $.fn.j
26 $log.append( "2nd loaded jQuery version (jq162): "
27 </script>
28
29 </body>
30 </html>
```

Demo

POWERED BY HERONOTE

A new version of this book is available!

---

# jQuery.noop()

Categories: [Utilities](#)

`jQuery.noop()`

Returns: `undefined`

**Description:** *An empty function.*

`jQuery.noop()`

version added: [1.4](#)

This method does not accept any arguments.

You can use this empty function when you wish to pass around a function that will do nothing.

This is useful for plugin authors who offer optional callbacks; in the case that no callback is given, something like `jQuery.noop` could execute.

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# jQuery.now()

Categories: [Utilities](#)

`jQuery.now()`

Returns: [Number](#)

**Description:** *Return a number representing the current time.*

`jQuery.now()`

version added: [1.4.3](#)

This method does not accept any arguments.

The `$.now()` method is a shorthand for the number returned by the expression `(new Date).getTime()`.

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# jQuery.param()

Categories: [Miscellaneous](#) > [Collection Manipulation](#) | [Forms](#) | [Ajax](#) > [Helper Functions](#)

`jQuery.param( obj )`

Returns: [String](#)

**Description:** *Create a serialized representation of an array or object, suitable for use in a URL query string or Ajax request.*

`jQuery.param( obj )`

version added: [1.2](#)

**obj**

Type: [Array](#) or [PlainObject](#)

An array or object to serialize.

`jQuery.param( obj, traditional )`

version added: [1.4](#)

**obj**

Type: [Array](#) or [PlainObject](#)

An array or object to serialize.

**traditional**

Type: [Boolean](#)

A Boolean indicating whether to perform a traditional "shallow" serialization.

This function is used internally to convert form element values into a serialized string representation (See [.serialize\(\)](#) for more information).

As of jQuery 1.3, the return value of a function is used instead of the function as a String.

As of jQuery 1.4, the `$.param()` method serializes deep objects recursively to accommodate modern scripting languages and frameworks such as PHP and Ruby on Rails. You can disable this functionality globally by setting `jQuery.ajaxSettings.traditional = true;`

If the object passed is in an Array, it must be an array of objects in the format returned by [.serializeArray\(\)](#)

```
1 | [{name: "first", value: "Rick"},  
2 |   {name: "last", value: "Astley"},  
3 |   {name: "job", value: "Rock Star"}]
```

**Note:** *Because some frameworks have limited ability to parse serialized arrays, developers should exercise caution when passing an `obj` argument that contains objects or arrays nested within another array.*

**Note:** *Because there is no universally agreed-upon specification for param strings, it is not possible to encode complex data structures using this method in a manner that works ideally across all languages supporting such input. Until such time that there is, the `$.param` method will remain in its current form.*

In jQuery 1.4, HTML5 input elements are also serialized.

We can display a query string representation of an object and a URI-decoded version of the same as follows:

```
1 | var myObject = {  
2 |   a: {  
3 |     one: 1,
```

```

4     two: 2,
5     three: 3
6   },
7   b: [1,2,3]
8 };
9 var recursiveEncoded = $.param(myObject);
10 var recursiveDecoded = decodeURIComponent($.param(myObject));
11
12 alert(recursiveEncoded);
13 alert(recursiveDecoded);

```

The values of `recursiveEncoded` and `recursiveDecoded` are alerted as follows:

`a%5Bone%5D=1&a%5Btwo%5D=2&a%5Bthree%5D=3&b%5B%5D=1&a[one]=1&a[two]=2&a[three]=3&b[]=1&b[]=2&b[]=3`

To emulate the behavior of `$.param()` prior to jQuery 1.4, we can set the `traditional` argument to `true`:

```

1   var myObject = {
2     a: {
3       one: 1,
4       two: 2,
5       three: 3
6     },
7     b: [1,2,3]
8   };
9   var shallowEncoded = $.param(myObject, true);
10  var shallowDecoded = decodeURIComponent(shallowEncoded);
11
12  alert(shallowEncoded);
13  alert(shallowDecoded);

```

The values of `shallowEncoded` and `shallowDecoded` are alerted as

follows:

`a=%5Bobject+Object%5D&b=1&b=2&b=3`

`a=[object+Object]&b=1&b=2&b=3`

## Examples:

**Example:** *Serialize a key/value object.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>div { color:red; }</style>
5   <script src="http://code.jquery.com/jquery-
6 </head>
7 <body>
8   <div id="results"></div>
9 <script>
10
11   var params = { width:1680, height:1050 };
12   var str = jQuery.param(params);
13   $("#results").text(str);
14 </script>
15
16 </body>
17 </html>
```

Demo

**Example:** *Serialize a few complex objects*

```
1 // <=1.3.2:
2 $.param({ a: [2,3,4] }) // "a=2&a=3&a=4"
3 // >=1.4:
4 $.param({ a: [2,3,4] }) // "a[]=2&a[]=3&a[]=4"
5
6 // <=1.3.2:
7 $.param({ a: { b:1,c:2 }, d: [3,4,{ e:5 }] })
8 // >=1.4:
```

```
9 | $.param({ a: { b:1,c:2 }, d: [3,4,{ e:5 }] }) // "a[b]=  
10 |
```

POWERED BY **HERONOTE**

A new version of this book is available!

---

# jQuery.parseHTML()

Categories: [Utilities](#)

`jQuery.parseHTML( data [, context ] [, keepScripts ] )`

Returns: [Array](#)

**Description:** *Parses a string into an array of DOM nodes.*

**jQuery.parseHTML( data [, context ] [, [version added: 1.8](#) keepScripts ] )**

**data**

Type: [String](#)

HTML string to be parsed

**context** (default: `document`)

Type: [Element](#)

DOM element to serve as the context in which the HTML fragment will be created

**keepScripts** (default: `false`)

Type: [Boolean](#)

A Boolean indicating whether to include scripts passed in the HTML string

`jQuery.parseHTML` uses a native DOM element creation function to convert the string to a set of DOM elements, which can then be inserted into the document.

## Example:

*Create an array of Dom nodes using an HTML string and insert it into a div.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
5 </head>
6 <body>
7
8 <div id="log">
9   <h3>Content:</h3>
10 </div>
11
12 <script>
13 var $log = $( "#log" ),
14     str = "hello, <b>my name is</b> jQuery.",
15     html = $.parseHTML( str ),
16     nodeName = [];
17
18 // Append the parsed HTML
19 $log.append( html );
20
21 // Gather the parsed HTML's node names
22 $.each( html, function( i, el ) {
23     nodeName[i] = "<li>" + el.nodeName + "</li>";
24 });
25
26 // Insert the node names
27 $log.append( "<h3>Node Names:</h3>" );
28 $( "<ol></ol>" )
29   .append( nodeName.join( "" ) )
30   .appendTo( $log );
```

```
31  
32 </script>  
33  
34 </body>  
35 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# jQuery.parseJSON()

Categories: [Utilities](#)

`jQuery.parseJSON( json )`

Returns: [Object](#)

**Description:** *Takes a well-formed JSON string and returns the resulting JavaScript object.*

`jQuery.parseJSON( json )`

version added: [1.4.1](#)

**json**

Type: [String](#)

The JSON string to parse.

Passing in a malformed JSON string may result in an exception being thrown. For example, the following are all malformed JSON strings:

`{test: 1}` (test does not have double quotes around it).

`{'test': 1}` ('test' is using single quotes instead of double quotes).

Additionally if you pass in nothing, an empty string, null, or undefined, 'null' will be returned from parseJSON. Where the browser provides a native implementation of `JSON.parse`, jQuery uses it to parse the string. For details on the JSON format, see <http://json.org/>.

## Example:

*Parse a JSON string.*

```
1 | var obj = jQuery.parseJSON( '{"name": "John"}' );  
2 | alert( obj.name === "John" );
```

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# jQuery.parseXML()

Categories: [Utilities](#)

`jQuery.parseXML( data )`

Returns: [XMLDocument](#)

**Description:** *Parses a string into an XML document.*

`jQuery.parseXML( data )`

version added: [1.5](#)

**data**

Type: [String](#)

a well-formed XML string to be parsed

`jQuery.parseXML` uses the native parsing function of the browser to create a valid XML Document. This document can then be passed to `jQuery` to create a typical jQuery object that can be traversed and manipulated.

## Example:

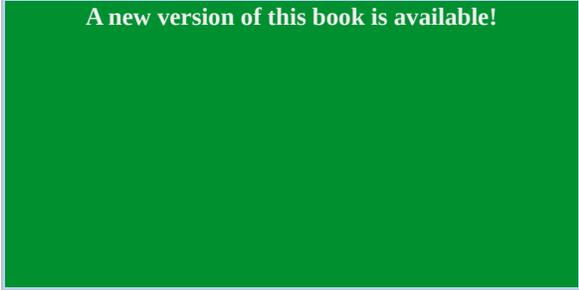
Create a jQuery object using an XML string and obtain the value of the title node.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-
5 </head>
6 <body>
7
8 <p id="someElement"></p>
9 <p id="anotherElement"></p>
10
11 <script>
12 var xml = "<rss version='2.0'><channel><title:
13   xmlDoc = $.parseXML( xml ),
14   $xml = $( xmlDoc ),
15   $title = $xml.find( "title" );
16
17 /* append "RSS Title" to #someElement */
18 $( "#someElement" ).append( $title.text() );
19
20 /* change the title to "XML Title" */
21 $title.text( "XML Title" );
22
23 /* append "XML Title" to #anotherElement */
24 $( "#anotherElement" ).append( $title.text() )
25 </script>
26
27 </body>
28 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

# jQuery.post()

Categories: [Ajax](#) > [Shorthand Methods](#)

`jQuery.post( url [, data ] [, success(data, textStatus, jqXHR) ] [, dataType ] )` Returns: [jqXHR](#)

**Description:** *Load data from the server using a HTTP POST request.*

**jQuery.post( url [, data ] [, success(data, textStatus, jqXHR) ] [, dataType ] )** version added: [1.0](#)

**url**

Type: [String](#)

A string containing the URL to which the request is sent.

**data**

Type: [PlainObject](#) or [String](#)

A plain object or string that is sent to the server with the request.

**success(data, textStatus, jqXHR)**

Type: [Function\(\)](#)

A callback function that is executed if the request succeeds.

**dataType**

Type: [String](#)

The type of data expected from the server. Default: Intelligent Guess (xml, json, script, text, html).

This is a shorthand Ajax function, which is equivalent to:

```
1 | $.ajax({
2 |   type: "POST",
3 |   url: url,
4 |   data: data,
5 |   success: success,
6 |   dataType: dataType
7 | });
```

The `success` callback function is passed the returned data, which will be an XML root element or a text string depending on the MIME type of the response. It is also passed the text status of the response.

As of **jQuery 1.5**, the `success` callback function is also passed a ["jqXHR" object](#) (in **jQuery 1.4**, it was passed the `XMLHttpRequest` object).

Most implementations will specify a success handler:

```
1 | $.post('ajax/test.html', function(data) {  
2 |     $(' .result').html(data);  
3 | });
```

This example fetches the requested HTML snippet and inserts it on the page.

Pages fetched with `POST` are never cached, so the `cache` and `ifModified` options in `jQuery.ajaxSetup()` have no effect on these requests.

## The jqXHR Object

As of **jQuery 1.5**, all of jQuery's Ajax methods return a superset of the `XMLHttpRequest` object. This jQuery XHR object, or "jqXHR," returned by `$.get()` implements the Promise interface, giving it all the properties, methods, and behavior of a Promise (see [Deferred object](#) for more information). The `jqXHR.done()` (for success), `jqXHR.fail()` (for error), and `jqXHR.always()` (for completion, whether success or error) methods take a function argument that is called when the request terminates. For information about the arguments this function receives, see the [jqXHR Object](#) section of the `$.ajax()` documentation.

The Promise interface also allows jQuery's Ajax methods, including `$.get()`, to chain multiple `.done()`, `.fail()`, and `.always()` callbacks on a single request, and even to assign these callbacks after the request may have completed. If the request is already complete, the callback is fired immediately.

```
1 // Assign handlers immediately after making the request
2 // and remember the jqxhr object for this request
3 var jqxhr = $.post("example.php", function() {
4     alert("success");
5 })
6 .done(function() { alert("second success"); })
7 .fail(function() { alert("error"); })
8 .always(function() { alert("finished"); });
9
10 // perform other work here ...
11
12 // Set another completion function for the request above
13 jqxhr.always(function(){ alert("second finished")
```

## Deprecation Notice

The `jqXHR.success()`, `jqXHR.error()`, and `jqXHR.complete()` callback methods introduced in jQuery 1.5 are **deprecated as of jQuery 1.8**. To prepare your code for their eventual removal, use `jqXHR.done()`, `jqXHR.fail()`, and `jqXHR.always()` instead.

## Additional Notes:

Due to browser security restrictions, most "Ajax" requests are subject to the [same origin policy](#); the request can not successfully retrieve data from a different domain, subdomain, or protocol.

If a request with `jQuery.post()` returns an error code, it will fail silently unless the script has also called the global `.ajaxError()` method. Alternatively, as of jQuery 1.5, the `.error()` method of the `jqXHR` object returned by `jQuery.post()` is also available for error handling.

## Examples:

**Example:** Request the test.php page, but ignore the return results.

```
1 | $.post("test.php");
```

**Example:** Request the test.php page and send some additional data along (while still ignoring the return results).

```
1 | $.post("test.php", { name: "John", time: "2pm"
```

**Example:** Pass arrays of data to the server (while still ignoring the return results).

```
1 | $.post("test.php", { 'choices[]': ["Jon", "Sus"
```

**Example:** Send form data using ajax requests

```
1 | $.post("test.php", $("#testform").serialize())
```

**Example:** Alert the results from requesting test.php (HTML or XML, depending on what was returned).

```
1 | $.post("test.php", function(data) {  
2 |     alert("Data Loaded: " + data);  
3 | });
```

**Example:** Alert the results from requesting test.php with an additional payload of data (HTML or XML, depending on what was returned).

```
1 | $.post("test.php", { name: "John", time: "2pm"
2 |   .done(function(data) {
3 |     alert("Data Loaded: " + data);
4 |   });
```

**Example:** Post to the test.php page and get content which has been returned in json format (<?php echo json\_encode(array("name"=>"John","time"=>"2pm")); ?>).

```
1 | $.post("test.php", { "func": "getNameAndTime"
2 |   function(data){
3 |     console.log(data.name); // John
4 |     console.log(data.time); // 2pm
5 |   }, "json");
```

**Example:** Post a form using ajax and put results in a div

```
1 | <!DOCTYPE html>
2 | <html>
3 | <head>
4 |   <script src="http://code.jquery.com/jquery-...
5 | </head>
6 | <body>
7 |   <form action="/" id="searchForm">
8 |     <input type="text" name="s" placeholder="S
9 |     <input type="submit" value="Search" />
```

```
10 </form>
11 <!-- the result of the search will be rendered inside
12 <div id="result"></div>
13
14 <script>
15 /* attach a submit handler to the form */
16 $("#searchForm").submit(function(event) {
17
18     /* stop form from submitting normally */
19     event.preventDefault();
20
21     /* get some values from elements on the page: */
22     var $form = $( this ),
23         term = $form.find( 'input[name="s"]' ).val(),
24         url = $form.attr( 'action' );
25
26     /* Send the data using post */
27     var posting = $.post( url, { s: term } );
28
29     /* Put the results in a div */
30     posting.done(function( data ) {
31         var content = $( data ).find( '#content' );
32         $( "#result" ).empty().append( content );
33     });
34 });
35 </script>
36
37 </body>
38 </html>
```

Demo

**A new version of this book is available!**

---

# jQuery.proxy()

Categories: [Events](#) > [Event Handler Attachment](#) | [Utilities](#)

**jQuery.proxy( function, context )**

Returns: [Function](#)

**Description:** *Takes a function and returns a new one that will always have a particular context.*

**jQuery.proxy( function, context )**

version added: [1.4](#)

**function**

Type: [Function\(\)](#)

The function whose context will be changed.

**context**

Type: [PlainObject](#)

The object to which the context (`this`) of the function should be set.

**jQuery.proxy( context, name )**

version added: [1.4](#)

**context**

Type: [PlainObject](#)

The object to which the context of the function should be set.

**name**

Type: [String](#)

The name of the function whose context will be changed (should be a property of the `context` object).

**jQuery.proxy( function, context [, additionalArguments ] )**

version added: [1.6](#)

**function**

Type: [Function\(\)](#)

The function whose context will be changed.

**context**

Type: [PlainObject](#)

The object to which the context (`this`) of the function should be set.

**additionalArguments**

Type: [Anything](#)

Any number of arguments to be passed to the function referenced in the `function` argument.

## jQuery.proxy( context, name [, additionalArguments ] )

version added: [1.6](#)

### context

Type: [PlainObject](#)

The object to which the context of the function should be set.

### name

Type: [String](#)

The name of the function whose context will be changed (should be a property of the `context` object).

### additionalArguments

Type: [Anything](#)

Any number of arguments to be passed to the function named in the `name` argument.

This method is most useful for attaching event handlers to an element where the context is pointing back to a different object. Additionally, jQuery makes sure that even if you bind the function returned from `jQuery.proxy()` it will still unbind the correct function if passed the original.

Be aware, however, that jQuery's event binding subsystem assigns a unique id to each event handling function in order to track it when it is used to specify the function to be unbound. The function represented by `jQuery.proxy()` is seen as a single function by the event subsystem, even when it is used to bind different contexts. To avoid unbinding the wrong handler, use a unique event namespace for binding and unbinding (e.g., `"click.myproxy1"`) rather than specifying the proxied function during unbinding.

**As of jQuery 1.6**, any number of additional arguments may be supplied to `$.proxy()`, and they will be passed to the function whose context will be changed.

## Examples:

**Example:** Change the context of functions bound to a click handler using the "function context" signature. Unbind the first handler after first click.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-...
5 </head>
6 <body>
7
8 <p><button type="button" id="test">Test</button>
9 <div id="log"></div>
10
11 <script>
12 var me = {
13   type: "zombie",
14   test: function(event) {
15     /* Without proxy, `this` would refer to the
16     /* use event.target to reference that element
17     var element = event.target;
18     $(element).css("background-color", "red");
19
20     /* With proxy, `this` refers to the me object
21     /* this function. */
22     $("#log").append( "Hello " + this.type + "
23     $("#test").off("click", this.test);
24   }
25 };
26
27 var you = {
28   type: "person",
29   test: function(event) {
```

```

30     $("#log").append( this.type + " " );
31   }
32 };
33
34 /* execute you.test() in the context of the `you` object
35 /* no matter where it is called */
36 /* i.e. the `this` keyword will refer to `you` */
37 var youClick = $.proxy( you.test, you );
38
39
40 /* attach click handlers to #test */
41 $("#test")
42
43   /* this === "zombie"; handler unbound after first click
44   .on( "click", $.proxy( me.test, me ) )
45
46   /* this === "person" */
47   .on( "click", youClick )
48
49   /* this === "zombie" */
50   .on( "click", $.proxy( you.test, me ) )
51
52   /* this === "<button> element" */
53   .on( "click", you.test );
54 </script>
55
56 </body>
57 </html>

```

## Demo

**Example:** Enforce the context of the function using the "context function name" signature. Unbind the handler after first click.

```
1 <!DOCTYPE html>
```

```

2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-latest.js"
5 </head>
6 <body>
7
8   <p><button id="test">Test</button></p>
9   <p id="log"></p>
10
11 <script>
12   var obj = {
13     name: "John",
14     test: function() {
15       $("#log").append( this.name );
16       $("#test").off("click", obj.test);
17     }
18   };
19
20   $("#test").on( "click", jQuery.proxy( obj, "test"
21 </script>
22
23 </body>
24 </html>

```

## Demo

**Example** · *Change the context of a function bound to the click handler,*

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-
5 </head>
6 <body>
7

```

```

8 <p><button type="button" id="test">Test</button>
9 <div id="log"></div>
10
11 <script>
12 var me = {
13     /* I'm a dog */
14     type: "dog",
15
16     /* Note that event comes *after* one and two */
17     test: function(one, two, event) {
18         $("#log")
19
20         /* `one` maps to `you`, the 1st additional */
21         /* argument in the $.proxy function call */
22         .append( "<h3>Hello " + one.type + " :</h3>"
23
24         /* the `this` keyword refers to `me` */
25         /* (the 2nd, context, argument of $.proxy) */
26         .append( "I am a " + this.type + ", " )
27
28         /* `two` maps to `they`, the 2nd additional */
29         /* argument in the $.proxy function call */
30         .append( "and they are " + two.type + "<br>"
31
32         /* the event type is "click" */
33         .append( "Thanks for " + event.type + "ing "
34
35         /* the clicked element is `event.target`, */
36         /* and its type is "button" */
37         .append( "the " + event.target.type + "."
38     }
39 };
40
41 var you = { type: "cat" };
42 var they = { type: "fish" };
43
44

```

```
45  /* Set up handler to execute me.test() in the context */
46  /* of `me`, with `you` and `they` as additional arguments */
47  var proxy = $.proxy( me.test, me, you, they );
48
49  $("#test")
50  .on( "click", proxy );
51  </script>
52
53  </body>
54  </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# jQuery.queue()

Categories: [Data](#) | [Utilities](#)

Show or manipulate the queue of functions to be executed on the matched element.

## Contents:

[jQuery.queue\( element \[, queueName \] \)](#)

[jQuery.queue\( element \[, queueName \] \)](#)

[jQuery.queue\( element, queueName, newQueue \)](#)

[jQuery.queue\( element, queueName, newQueue \)](#)

[jQuery.queue\( element, queueName, callback\(\) \)](#)

---

`jQuery.queue( element [, queueName ] )`

Returns: `Array`

**Description:** *Show the queue of functions to be executed on the matched element.*

**jQuery.queue( element [, queueName ] )** version added: [1.3](#)

**element**

Type: `Element`

A DOM element to inspect for an attached queue.

**queueName**

Type: `String`

A string containing the name of the queue. Defaults to `fx`, the standard effects queue.

**Note:** This is a low-level method, you should probably use `.queue()` instead.

## Example:

*Show the length of the queue.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <style>div { margin:3px; width:40px; height
5          position:absolute; left:0px; top:30px
6          background:green; display:none; }
7      div.newcolor { background:blue; }
8      span { color:red; } </style>
9      <script src="http://code.jquery.com/jquery-
10 </head>
11 <body>
12     <button id="show">Show Length of Queue</but
13     <span></span>
14     <div></div>
15
16 <script>
17     $( "#show" ).click( function () {
18         var n = jQuery.queue( $( "div" )[0], "fx" )
19         $( "span" ).text( "Queue length is: " + n.le
20     });
21     function runIt() {
22         $( "div" ).show( "slow" );
23         $( "div" ).animate( {left: '+=200' }, 2000 );
24         $( "div" ).slideToggle( 1000 );
25         $( "div" ).slideToggle( "fast" );
26         $( "div" ).animate( {left: '-=200' }, 1500 );
27         $( "div" ).hide( "slow" );
28         $( "div" ).show( 1200 );
29         $( "div" ).slideUp( "normal", runIt );
30     }
31     runIt();
```

```
32     </script>
```

```
33
```

```
34 </body>
```

```
35 </html>
```

Demo

`jQuery.queue( element, queueName, newQueue )`

Returns: `jQuery`

**Description:** *Manipulate the queue of functions to be executed on the matched element.*

**jQuery.queue( element, queueName, newQueue )** version added: [1.3](#)

**element**

Type: [Element](#)

A DOM element where the array of queued functions is attached.

**queueName**

Type: [String](#)

A string containing the name of the queue. Defaults to `fx`, the standard effects queue.

**newQueue**

Type: [Array](#)

An array of functions to replace the current queue contents.

**jQuery.queue( element, queueName, callback() )** version added: [1.3](#)

**element**

Type: [Element](#)

A DOM element on which to add a queued function.

**queueName**

Type: [String](#)

A string containing the name of the queue. Defaults to `fx`, the standard effects queue.

**callback()**

Type: [Function\(\)](#)

The new function to add to the queue.

**Note:** This is a low-level method, you should probably use `.queue()` instead.

Every element can have one or more queues of functions attached to it by jQuery. In most applications, only one queue (called `fx`) is used. Queues allow a sequence of actions to be called on an element asynchronously, without halting program execution.

The `jQuery.queue()` method allows us to directly manipulate this queue of functions. Calling `jQuery.queue()` with a callback is particularly useful; it allows us to place a new function at the end of the queue.

Note that when adding a function with `jQuery.queue()`, we should ensure that `jQuery.dequeue()` is eventually called so that the next function in line executes.

## Examples:

**Example:** *Queue a custom function.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     div { margin:3px; width:40px; height:40px;
6           position:absolute; left:0px; top:30px;
7           background:green; display:none; }
8     div.newcolor { background:blue; }
9   </style>
10  <script src="http://code.jquery.com/jquery-
11 </head>
12 <body>
13   Click here...
14   <div></div>
15
16  <script>
17    $(document.body).click(function () {
18      $("div").show("slow");
19      $("div").animate({left:'+=200'}, 2000);
20      jQuery.queue( $("div")[0], "fx", function
21        $(this).addClass("newcolor");
22        jQuery.dequeue( this );
23      });
24      $("div").animate({left:'-=200'}, 500);
25      jQuery.queue( $("div")[0], "fx", function
26        $(this).removeClass("newcolor");
27        jQuery.dequeue( this );
28      });
29      $("div").slideUp();
30    });</script>
31
```

```
32 </body>
33 </html>
```

Demo

**Example:** *Set a queue array to delete the queue.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <style>
5   div { margin:3px; width:40px; height:40px;
6         position:absolute; left:0px; top:30px
7         background:green; display:none; }
8   div.newcolor { background:blue; }
9 </style>
10 <script src="http://code.jquery.com/jquery-
11 </head>
12 <body>
13
14 <button id="start">Start</button>
15 <button id="stop">Stop</button>
16 <div></div>
17 <script>
18   $("#start").click(function () {
19     $("#div").show("slow");
20     $("#div").animate({left:'+=200'}, 5000);
21     jQuery.queue( $("#div")[0], "fx", function
22       $(this).addClass("newcolor");
23       jQuery.dequeue( this );
24     });
25     $("#div").animate({left:'-=200'}, 1500);
26     jQuery.queue( $("#div")[0], "fx", function
27       $(this).removeClass("newcolor");
28       jQuery.dequeue( this );
29     });
```

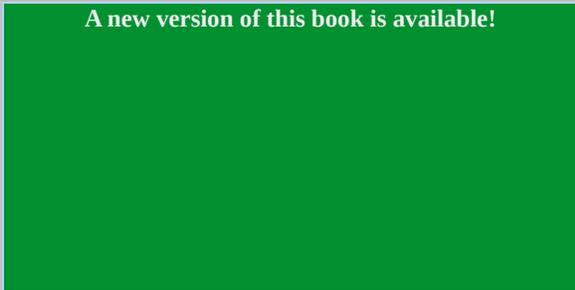
```
30     $("div").slideUp();
31     });
32     $("#stop").click(function () {
33         jQuery.queue( $("div")[0], "fx", [] );
34         $("div").stop();
35     });
36 </script>
37
38 </body>
39 </html>
```



Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

# jQuery.removeData()

Categories: [Data](#) | [Utilities](#)

`jQuery.removeData( element [, name ] )` Returns: [jQuery](#)

**Description:** *Remove a previously-stored piece of data.*

`jQuery.removeData( element [, name ] )` version added: [1.2.3](#)

**element**

Type: [Element](#)

A DOM element from which to remove data.

**name**

Type: [String](#)

A string naming the piece of data to remove.

**Note:** This is a low-level method, you should probably use `jQuery.removeData()` instead.

The `jQuery.removeData()` method allows us to remove values that were previously set using `jQuery.data()`. When called with the name of a key, `jQuery.removeData()` deletes that particular value; when called with no arguments, all values are removed.

## Example:

*Set a data store for 2 names then remove one of them.*

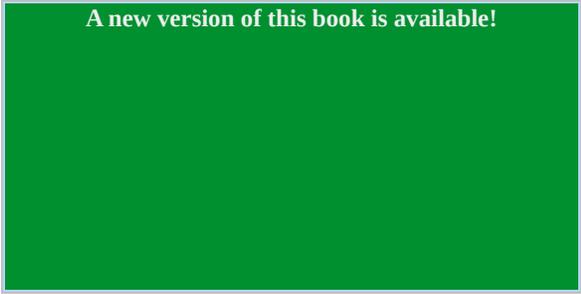
```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     div { margin:2px; color:blue; }
6     span { color:red; }
7   </style>
8   <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
9 </head>
10 <body>
11   <div>value1 before creation: <span></span><br>
12     <div>value1 after creation: <span></span><br>
13     <div>value1 after removal: <span></span><br>
14     <div>value2 after removal: <span></span><br>
15   <script>
16     var div = $("div")[0];
17     $("span:eq(0)").text(" " + $(div).data("test1"));
18     jQuery.data(div, "test1", "VALUE-1");
19     jQuery.data(div, "test2", "VALUE-2");
20     $("span:eq(1)").text(" " + jQuery.data(div, "test2"));
21     jQuery.removeData(div, "test1");
22     $("span:eq(2)").text(" " + jQuery.data(div, "test2"));
23     $("span:eq(3)").text(" " + jQuery.data(div, "test2"));
24   </script>
25 </body>
26 </html>
```

Demo

---

POWERED BY HERONOTE

A new version of this book is available!



---

# jQuery.sub()

Categories: [Core](#)

jQuery.sub()

Returns: [jQuery](#)

version deprecated: [1.7](#), removed: [1.9](#)

**Description:** *Creates a new copy of jQuery whose properties and methods can be modified without affecting the original jQuery object.*

jQuery.sub()

version added: [1.5](#)

This method does not accept any arguments.

**This method is deprecated as of jQuery 1.7 and will be moved to a plugin in jQuery 1.8.**

There are two specific use cases for which jQuery.sub() was created. The first was for providing a painless way of overriding jQuery methods without completely destroying the original methods and another was for helping to do encapsulation and basic namespacing for jQuery plugins.

Note that jQuery.sub() doesn't attempt to do any sort of isolation - that's not its intention. All the methods on the sub'd version of jQuery will still point to the original jQuery (events bound and triggered will still be through the main jQuery, data will be bound to elements through the main jQuery, Ajax queries and events will run through the main jQuery, etc.).

Note that if you're looking to use this for plugin development you should first *strongly* consider using something like the jQuery UI widget factory which manages both state and plugin sub-methods. [Some examples of using the jQuery UI widget factory](#) to build a plugin.

The particular use cases of this method can be best described through some examples.

## Examples:

**Example:** Adding a method to a jQuery sub so that it isn't exposed externally:

```
1 (function(){
2   var sub$ = jQuery.sub();
3
4   sub$.fn.myCustomMethod = function(){
5     return 'just for me';
6   };
7
8   sub$(document).ready(function() {
9     sub$('body').myCustomMethod() // 'just fo
10  });
11 })();
12
13 typeof jQuery('body').myCustomMethod // undef.
```

**Example:** Override some jQuery methods to provide new functionality.

```
1 (function() {
2   var myjQuery = jQuery.sub();
3
4   myjQuery.fn.remove = function() {
5     // New functionality: Trigger a remove event
6     this.trigger("remove");
7
8     // Be sure to call the original jQuery remove
9     return jQuery.fn.remove.apply( this, arguments
10  };
11
```

```

12 myjQuery(function($) {
13     $(".menu").click(function() {
14         $(this).find(".submenu").remove();
15     });
16
17     // A new remove event is now triggered from this copy
18     $(document).on("remove", function(e) {
19         $(e.target).parent().hide();
20     });
21 });
22 })();
23
24 // Regular jQuery doesn't trigger a remove event when removed
25 // This functionality is only contained within the modified jQuery

```

**Example** Create a plugin that returns plugin-specific methods.

```

1 (function() {
2     // Create a new copy of jQuery using sub()
3     var plugin = jQuery.sub();
4
5     // Extend that copy with the new plugin methods
6     plugin.fn.extend({
7         open: function() {
8             return this.show();
9         },
10        close: function() {
11            return this.hide();
12        }
13    });
14
15    // Add our plugin to the original jQuery
16    jQuery.fn.myplugin = function() {
17        this.addClass("plugin");

```

```
18
19     // Make sure our plugin returns our special plugin
20     return plugin( this );
21 };
22 })();
23
24 $(document).ready(function() {
25     // Call the plugin, open method now exists
26     $('#main').myplugin().open();
27
28     // Note: Calling just $("#main").open() won't work as
29 });
```

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# jQuery.support

Categories: [Properties](#) > [Properties of the Global jQuery Object](#) | [Utilities](#)

jQuery.support

Returns: [Object](#)

**Description:** *A collection of properties that represent the presence of different browser features or bugs. Primarily intended for jQuery's internal use; specific properties may be removed when they are no longer needed internally to improve page startup performance.*

jQuery.support

version added: [1.3](#)

Rather than using `$.browser` to detect the current user agent and alter the page presentation based on which browser is running, it is a good practice to use **feature detection**. To make this process simpler, jQuery performs many such tests and sets properties of the `jQuery.support` object.

Since jQuery requires these tests internally, they must be performed on *every* page load. Although some of these properties are documented below, they are not subject to a long deprecation/removal cycle and may be removed once internal jQuery code no longer needs them.

*Following are a few resources that explain how feature detection works:*

<http://peter.michaux.ca/articles/feature-detection-state-of-the-art-browser-scripting>

[http://www.jibbering.com/faq/faq\\_notes/not\\_browser\\_detect.html](http://www.jibbering.com/faq/faq_notes/not_browser_detect.html)

<http://yura.thinkweb2.com/cft/>

For your own project's feature-detection needs, we strongly recommend the use of an external library such as [Modernizr](#) instead of dependency on properties in `jQuery.support`.

The tests included in `jQuery.support` are as follows:

`ajax` is equal to true if a browser is able to create an `XMLHttpRequest` object.

`boxModel` is equal to true if the page is rendering according to the [W3C CSS Box Model](#) (is currently false in IE 6 and 7 when they are in Quirks Mode). This property is null until document ready occurs.

`changeBubbles` is equal to true if the change event bubbles up the DOM tree, as required by the [W3C DOM event model](#). (It is currently false in IE, and jQuery simulates bubbling).

`checkClone` is equal to true if a browser correctly clones the checked state of radio buttons or checkboxes in document fragments.

`checkOn` is equal to true if the value of a checkbox defaults to "on" when no value is specified.

`cors` is equal to true if a browser can create an `XMLHttpRequest` object and if that `XMLHttpRequest` object has a `withCredentials` property. To enable cross-domain requests in environments that do not support cors yet but do allow cross-domain XHR requests (windows gadget, etc), set `$.support.cors = true;`.  
[CORS WD](#)

`cssFloat` is equal to true if the name of the property containing the CSS float value is `.cssFloat`, as defined in the [CSS Spec](#). (It is currently false in IE, it uses `styleFloat` instead).

`hrefNormalized` is equal to true if the `.getAttribute()` method retrieves the `href` attribute of elements unchanged, rather than normalizing it to a fully-qualified URL. (It is currently false in IE, the URLs are normalized).  
[DOM I3 spec](#)

`htmlSerialize` is equal to true if the browser is able to serialize/insert `<link>` elements using the `.innerHTML` property of elements. (is currently false in IE).  
[HTML5 WD](#)

`leadingWhitespace` is equal to true if the browser inserts content with `.innerHTML` exactly as provided—specifically, if leading whitespace characters are preserved. (It is currently false in IE 6-8).  
[HTML5 WD](#)

`noCloneChecked` is equal to true if cloned DOM elements copy

over the state of the `.checked` expando. (It is currently false in IE). (Added in jQuery 1.5.1)

`noCloneEvent` is equal to true if cloned DOM elements are created without event handlers (that is, if the event handlers on the source element are not cloned). (It is currently false in IE).  
[DOM I2 spec](#)

`opacity` is equal to true if a browser can properly interpret the opacity style property. (It is currently false in IE, it uses alpha filters instead).  
[CSS3 spec](#)

`optDisabled` is equal to true if option elements within disabled select elements are not automatically marked as disabled.  
[HTML5 WD](#)

`optSelected` is equal to true if an `<option>` element that is selected by default has a working `selected` property.  
[HTML5 WD](#)

`scriptEval()` is equal to true if inline scripts are automatically evaluated and executed when inserted into the document using standard DOM manipulation methods such as `.appendChild()` and `.createTextNode()`. (It is currently false in IE, it uses `.text` to insert executable scripts).

**Note: No longer supported; removed in jQuery 1.6. Prior to jQuery 1.5.1, the `scriptEval()` method was the static `scriptEval` property. The change to a method allowed the test to be deferred until first use to prevent content security policy inline-script violations.**

[HTML5 WD](#)

`style` is equal to true if inline styles for an element can be accessed through the DOM attribute called style, as required by the DOM Level 2 specification. In this case, `.getAttribute('style')` can retrieve this value; in Internet Explorer, `.cssText` is used for this purpose.

[DOM I2 Style spec](#)

`submitBubbles` is equal to true if the submit event bubbles up the DOM tree, as required by the [W3C DOM event model](#). (It is currently false in IE, and jQuery simulates bubbling).

`tbody` is equal to true if an empty `<table>` element can exist without a `<tbody>` element. According to the HTML specification, this sub-element is optional, so the property should be true in a fully-compliant browser. If false, we must account for the

possibility of the browser injecting `<tbody>` tags implicitly. (It is currently false in IE, which automatically inserts `tbody` if it is not present in a string assigned to `innerHTML`).

[HTML5 spec](#)

## Examples:

**Example:** *Returns the box model for the iframe.*

```
1 | <!DOCTYPE html>
2 | <html>
3 | <head>
4 |   <style>
5 |     p { color:blue; margin:20px; }
6 |     span { color:red; }
7 |   </style>
8 |   <script src="http://code.jquery.com/jquery-
9 | </head>
10 | <body>
11 |   <p>
12 |   </p>
13 | <script>
14 |   $("p").html("This frame uses the W3C box mode.
15 |               jQuery.support.boxModel + "</span:
16 | </script>
17 |
18 | </body>
19 | </html>
```

Demo

**Example:** *Returns false if the page is in QuirksMode in Internet Explorer*

```
1 | jQuery.support.boxModel
```

**Result:**

1 | false

POWERED BY HERONOTE

A new version of this book is available!

---

# jQuery.trim()

Categories: [Utilities](#)

`jQuery.trim( str )`

Returns: [String](#)

**Description:** *Remove the whitespace from the beginning and end of a string.*

`jQuery.trim( str )`

version added: [1.0](#)

**str**

Type: [String](#)

The string to trim.

The `$.trim()` function removes all newlines, spaces (including non-breaking spaces), and tabs from the beginning and end of the supplied string. If these whitespace characters occur in the middle of the string, they are preserved.

## Examples:

**Example:** Remove the two white spaces at the start and at the end of the string.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-1.10.2.min.js"></script>
5 </head>
6 <body>
7
8   <pre id="original"></pre>
9   <pre id="trimmed"></pre>
10
11 <script>
12   var str = "   lots of spaces before and after   ";
13   $("#original").html("Original String: '" + str + "'");
14   $("#trimmed").html("$.trim()'ed: '" + $.trim(str) + "'");
15 </script>
16
17 </body>
18 </html>
```

Demo

**Example:** Remove the two white spaces at the start and at the end of the string.

```
1 | $.trim("   hello, how are you?   ");
```

**Result:**

1 | "hello, how are you?"

POWERED BY **HERONOTE**

A new version of this book is available!

---

# jQuery.type()

Categories: [Utilities](#)

`jQuery.type( obj )`

Returns: [String](#)

**Description:** *Determine the internal JavaScript `[[Class]]` of an object.*

`jQuery.type( obj )`

version added: [1.4.3](#)

**obj**

Type: [PlainObject](#)

Object to get the internal JavaScript `[[Class]]` of.

A number of techniques are used to determine the exact return value for an object. The `[[Class]]` is determined as follows:

If the object is undefined or null, then "undefined" or "null" is returned accordingly.

```
jQuery.type( undefined ) === "undefined"
```

```
jQuery.type() === "undefined"
```

```
jQuery.type( window.notDefined ) === "undefined"
```

```
jQuery.type( null ) === "null"
```

If the object has an internal `[[Class]]` equivalent to one of the browser's built-in objects, the associated name is returned.

[\(More details about this technique.\)](#)

```
jQuery.type( true ) === "boolean"
```

```
jQuery.type( 3 ) === "number"
```

```
jQuery.type( "test" ) === "string"
```

```
jQuery.type( function(){} ) === "function"
```

```
jQuery.type( [] ) === "array"
```

```
jQuery.type( new Date() ) === "date"
```

```
jQuery.type( new Error() ) === "error" // as of jQuery 1.9
```

```
jQuery.type( /test/ ) === "regexp"
```

Everything else returns "object" as its type.



## Example:

*Find out if the parameter is a RegExp.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-
5 </head>
6 <body>
7   Is it a RegExp? <b></b>
8   <script>$("b").append( "" + jQuery.type(/test.
9
10 </body>
11 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# jQuery.unique()

Categories: [Utilities](#)

`jQuery.unique( array )`

Returns: [Array](#)

**Description:** *Sorts an array of DOM elements, in place, with the duplicates removed. Note that this only works on arrays of DOM elements, not strings or numbers.*

`jQuery.unique( array )`

version added: [1.1.3](#)

**array**

Type: [Array](#)

The Array of DOM elements.

The `$.unique()` function searches through an array of objects, sorting the array, and removing any duplicate nodes. This function only works on plain JavaScript arrays of DOM elements, and is chiefly used internally by jQuery.

As of jQuery 1.4 the results will always be returned in document order.

## Example:

*Removes any duplicate elements from the array of divs.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     div { color:blue; }
6   </style>
7   <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
8 </head>
9 <body>
10  <div>There are 6 divs in this document.</div>
11  <div></div>
12  <div class="dup"></div>
13  <div class="dup"></div>
14
15  <div class="dup"></div>
16  <div></div>
17 <script>
18
19   var divs = $("div").get(); // unique() must
20
21   // add 3 elements of class dup too (they are
22   divs = divs.concat($(".dup").get());
23   $("div:eq(1)").text("Pre-unique there are "
24
25   divs = jQuery.unique(divs);
26   $("div:eq(2)").text("Post-unique there are
27                       .css("color", "red");
28 </script>
29
30 </body>
```

31 | `</html>`

Demo

POWERED BY **HERONOTE**

A new version of this book is available!

---

# jQuery.when()

Categories: [Core](#) | [Deferred Object](#)

`jQuery.when( deferreds )`

Returns: [Promise](#)

**Description:** Provides a way to execute callback functions based on one or more objects, usually [Deferred](#) objects that represent asynchronous events.

`jQuery.when( deferreds )`

version added: [1.5](#)

**deferreds**

Type: [Deferred](#)

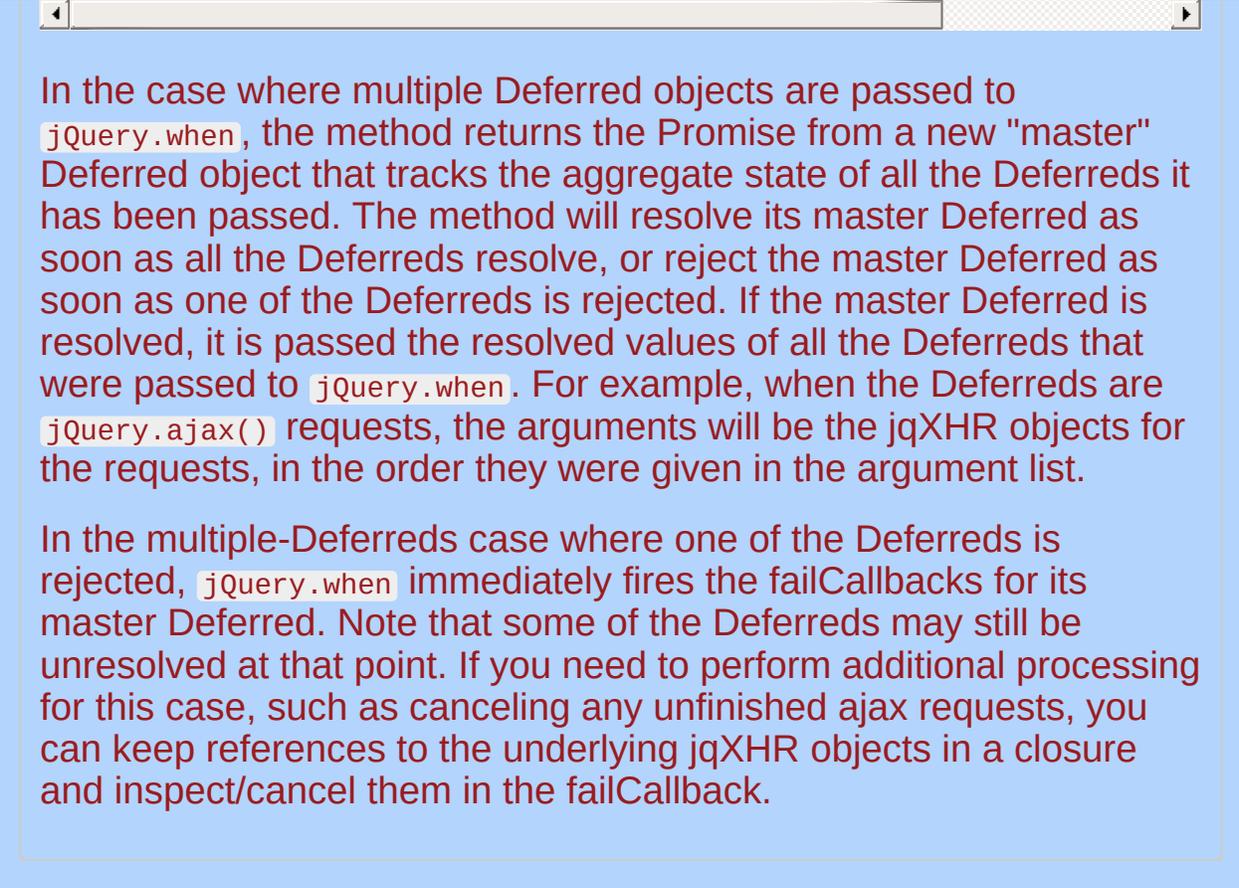
One or more Deferred objects, or plain JavaScript objects.

If a single Deferred is passed to `jQuery.when`, its Promise object (a subset of the Deferred methods) is returned by the method. Additional methods of the Promise object can be called to attach callbacks, such as `deferred.then`. When the Deferred is resolved or rejected, usually by the code that created the Deferred originally, the appropriate callbacks will be called. For example, the `jqXHR` object returned by `jQuery.ajax()` is a Promise and can be used this way:

```
1 | $.when( $.ajax("test.aspx") ).then(function(d
2 |     alert( jqXHR.status ); // alerts 200
3 | });
```

If a single argument is passed to `jQuery.when` and it is not a Deferred or a Promise, it will be treated as a resolved Deferred and any doneCallbacks attached will be executed immediately. The doneCallbacks are passed the original argument. In this case any failCallbacks you might set are never called since the Deferred is never rejected. For example:

```
1 | $.when( { testing: 123 } ).done(
2 |     function(x) { alert(x.testing); } /* alert
3 | );
```



In the case where multiple Deferred objects are passed to `jQuery.when`, the method returns the Promise from a new "master" Deferred object that tracks the aggregate state of all the Deferreds it has been passed. The method will resolve its master Deferred as soon as all the Deferreds resolve, or reject the master Deferred as soon as one of the Deferreds is rejected. If the master Deferred is resolved, it is passed the resolved values of all the Deferreds that were passed to `jQuery.when`. For example, when the Deferreds are `jQuery.ajax()` requests, the arguments will be the jqXHR objects for the requests, in the order they were given in the argument list.

In the multiple-Deferreds case where one of the Deferreds is rejected, `jQuery.when` immediately fires the failCallbacks for its master Deferred. Note that some of the Deferreds may still be unresolved at that point. If you need to perform additional processing for this case, such as canceling any unfinished ajax requests, you can keep references to the underlying jqXHR objects in a closure and inspect/cancel them in the failCallback.

## Examples:

**Example:** Execute a function after two ajax requests are successful (See the `jQuery.ajax()` documentation for a complete description of success and error cases for an ajax request).

```
1 | $.when($.ajax("/page1.php"), $.ajax("/page2.php"))  
2 |     /* a1 and a2 are arguments resolved for the  
3 |     page1 and page2 ajax requests, respectively  
4 |     var jqXHR = a1[2]; /* arguments are [ "success",  
5 |     if ( /Whip It/.test(jqXHR.responseText) ) {  
6 |         alert("First page has 'Whip It' somewhere.  
7 |     }  
8 | });
```

**Example:** Execute the function `myFunc` when both ajax requests are successful, or `myFailure` if either one has an error.

```
1 | $.when($.ajax("/page1.php"), $.ajax("/page2.php"))  
2 |     .then(myFunc, myFailure);
```

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# .keydown()

Categories: [Events](#) > [Keyboard Events](#)

`.keydown( handler(eventObject) )`

Returns: [jQuery](#)

**Description:** *Bind an event handler to the "keydown" JavaScript event, or trigger that event on an element.*

**.keydown( handler(eventObject) )**

version added: [1.0](#)

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute each time the event is triggered.

**.keydown( [eventData ],  
handler(eventObject) )**

version added: [1.4.3](#)

**eventData**

Type: [PlainObject](#)

An object containing data that will be passed to the event handler.

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute each time the event is triggered.

**.keydown()**

version added: [1.0](#)

This method does not accept any arguments.

This method is a shortcut for `.on('keydown', handler)` in the first and second variations, and `.trigger('keydown')` in the third.

The `keydown` event is sent to an element when the user first presses a key on the keyboard. It can be attached to any element, but the event is only sent to the element that has the focus. Focusable elements can vary between browsers, but form elements can always get focus so are reasonable candidates for this event type.

For example, consider the HTML:

1 | `<form>`

```
2 | <input id="target" type="text" value="Hello there"
3 | </form>
4 | <div id="other">
5 |   Trigger the handler
6 | </div>
```

The event handler can be bound to the input field:

```
1 | $('#target').keydown(function() {
2 |   alert('Handler for .keydown() called. ');
3 | });
```

Now when the insertion point is inside the field, pressing a key displays the alert:

Handler for .keydown() called.

To trigger the event manually, apply `.keydown()` without an argument:

```
1 | $('#other').click(function() {
2 |   $('#target').keydown();
3 | });
```

After this code executes, clicks on Trigger the handler will also alert the message.

If key presses anywhere need to be caught (for example, to implement global shortcut keys on a page), it is useful to attach this behavior to the `document` object. Because of event bubbling, all key presses will make their way up the DOM to the `document` object unless explicitly stopped.

To determine which key was pressed, examine the [event object](#) that is passed to the handler function. While browsers use differing properties to store this information, jQuery normalizes the `.which` property so you can reliably use it to retrieve the key code. This

code corresponds to a key on the keyboard, including codes for special keys such as arrows. For catching actual text entry, `.keypress()` may be a better choice.

## Example:

*Show the event object for the keydown handler when a key is pressed in the input.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5   fieldset { margin-bottom: 1em; }
6   input { display: block; margin-bottom: .25em;
7   #print-output {
8     width: 100%;
9   }
10  .print-output-line {
11    white-space: pre;
12    padding: 5px;
13    font-family: monaco, monospace;
14    font-size: .7em;
15  }
16
17 </style>
18   <script src="http://code.jquery.com/jquery-
19 </head>
20 <body>
21   <form>
22     <fieldset>
23       <label for="target">Type Something:</label>
24       <input id="target" type="text" />
25     </fieldset>
26   </form>
27   <button id="other">
28     Trigger the handler
29   </button>
30   <script type="text/javascript" src="/resource
```

```
31 <script>
32 var xTriggered = 0;
33 $('#target').keydown(function(event) {
34     if (event.which == 13) {
35         event.preventDefault();
36     }
37     xTriggered++;
38     var msg = 'Handler for .keydown() called ' + xTriggered;
39     $.print(msg, 'html');
40     $.print(event);
41 });
42
43 $('#other').click(function() {
44     $('#target').keydown();
45 });</script>
46
47 </body>
48 </html>
```

Demo

POWERED BY HERONOTE

A new version of this book is available!

---

# .keypress()

Categories: [Events](#) > [Keyboard Events](#)

`.keypress( handler(eventObject) )`

Returns: [jQuery](#)

**Description:** *Bind an event handler to the "keypress" JavaScript event, or trigger that event on an element.*

`.keypress( handler(eventObject) )`

version added: [1.0](#)

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute each time the event is triggered.

`.keypress( [eventData ],  
handler(eventObject) )`

version added: [1.4.3](#)

**eventData**

Type: [PlainObject](#)

An object containing data that will be passed to the event handler.

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute each time the event is triggered.

`.keypress()`

version added: [1.0](#)

This method does not accept any arguments.

**Note:** as the `keypress` event isn't covered by any official specification, the actual behavior encountered when using it may differ across browsers, browser versions, and platforms.

This method is a shortcut for `.on('keypress', handler)` in the first two variations, and `.trigger('keypress')` in the third.

The `keypress` event is sent to an element when the browser registers keyboard input. This is similar to the `keydown` event, except in the case of key repeats. If the user presses and holds a key, a `keydown` event is triggered once, but separate `keypress` events are triggered for each inserted character. In addition, modifier keys (such as Shift) trigger `keydown` events but not `keypress` events.

A `keypress` event handler can be attached to any element, but the event is only sent to the element that has the focus. Focusable elements can vary between browsers, but form elements can always get focus so are reasonable candidates for this event type.

For example, consider the HTML:

```
1 | <form>
2 |   <fieldset>
3 |     <input id="target" type="text" value="Hello" />
4 |   </fieldset>
5 | </form>
6 | <div id="other">
7 |   Trigger the handler
8 | </div>
```

The event handler can be bound to the input field:

```
1 | $("#target").keypress(function() {
2 |   alert("Handler for .keypress() called.");
3 | });
```

Now when the insertion point is inside the field, pressing a key displays the alert:

Handler for `.keypress()` called.

The message repeats if the key is held down. To trigger the event manually, apply `.keypress()` without an argument::

```
1 | $('#other').click(function() {
2 |   $("#target").keypress();
3 | });
```

After this code executes, clicks on Trigger the handler will also alert the message.

If key presses anywhere need to be caught (for example, to implement global shortcut keys on a page), it is useful to attach this behavior to the `document` object. Because of event bubbling, all key presses will make their way up the DOM to the `document` object unless explicitly stopped.

To determine which character was entered, examine the `event` object that is passed to the handler function. While browsers use differing properties to store this information, jQuery normalizes the `.which` property so you can reliably use it to retrieve the character code.

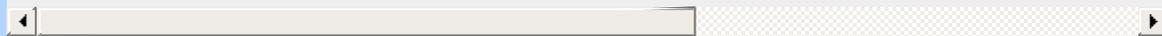
Note that `keydown` and `keyup` provide a code indicating which key is pressed, while `keypress` indicates which character was entered. For example, a lowercase "a" will be reported as 65 by `keydown` and `keyup`, but as 97 by `keypress`. An uppercase "A" is reported as 65 by all events. Because of this distinction, when catching special keystrokes such as arrow keys, `.keydown()` or `.keyup()` is a better choice.

## Example:

Show the event object when a key is pressed in the input. Note: This demo relies on a simple `$print()` plugin (<http://ani.innerv.com/resources/events.js>) for the event object's output.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     fieldset { margin-bottom: 1em; }
6     input { display: block; margin-bottom: .25em; }
7     #print-output {
8       width: 100%;
9     }
10    .print-output-line {
11      white-space: pre;
12      padding: 5px;
13      font-family: monaco, monospace;
14      font-size: .7em;
15    }
16
17  </style>
18  <script src="http://code.jquery.com/jquery-
19 </head>
20 <body>
21   <form>
22     <fieldset>
23       <label for="target">Type Something:</label>
24       <input id="target" type="text" />
25     </fieldset>
26   </form>
27   <button id="other">
28     Trigger the handler
```

```
29 </button>
30 <script src="http://api.jquery.com/resources/events.js">
31
32 <script>
33 var xTriggered = 0;
34 $("#target").keypress(function(event) {
35     if ( event.which == 13 ) {
36         event.preventDefault();
37     }
38     xTriggered++;
39     var msg = "Handler for .keypress() called " + xTriggered;
40     $.print( msg, "html" );
41     $.print( event );
42 });
43
44 $("#other").click(function() {
45     $("#target").keypress();
46 });</script>
47
48 </body>
49 </html>
```



Demo

POWERED BYHERONOTE

A new version of this book is available!

---

# .keyup()

Categories: [Events](#) > [Keyboard Events](#)

`.keyup( handler(eventObject) )`

Returns: [jQuery](#)

**Description:** *Bind an event handler to the "keyup" JavaScript event, or trigger that event on an element.*

**.keyup( handler(eventObject) )**

version added: [1.0](#)

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute each time the event is triggered.

**.keyup( [eventData ],  
handler(eventObject) )**

version added: [1.4.3](#)

**eventData**

Type: [PlainObject](#)

An object containing data that will be passed to the event handler.

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute each time the event is triggered.

**.keyup()**

version added: [1.0](#)

This method does not accept any arguments.

This method is a shortcut for `.on('keyup', handler)` in the first two variations, and `.trigger('keyup')` in the third.

The `keyup` event is sent to an element when the user releases a key on the keyboard. It can be attached to any element, but the event is only sent to the element that has the focus. Focusable elements can vary between browsers, but form elements can always get focus so are reasonable candidates for this event type.

For example, consider the HTML:

1 | `<form>`

```
2 | <input id="target" type="text" value="Hello there"  
3 | </form>  
4 | <div id="other">  
5 |   Trigger the handler  
6 | </div>
```

The event handler can be bound to the input field:

```
1 | $('#target').keyup(function() {  
2 |   alert('Handler for .keyup() called.');
```

Now when the insertion point is inside the field and a key is pressed and released, the alert is displayed:

Handler for .keyup() called.

To trigger the event manually, apply `.keyup()` without arguments:

```
1 | $('#other').click(function() {  
2 |   $('#target').keyup();  
3 | });
```

After this code executes, clicks on Trigger the handler will also alert the message.

If key presses anywhere need to be caught (for example, to implement global shortcut keys on a page), it is useful to attach this behavior to the `document` object. Because of event bubbling, all key presses will make their way up the DOM to the `document` object unless explicitly stopped.

To determine which key was pressed, examine the event object that is passed to the handler function. While browsers use differing properties to store this information, jQuery normalizes the `.which` property so you can reliably use it to retrieve the key code. This

code corresponds to a key on the keyboard, including codes for special keys such as arrows. For catching actual text entry, `.keypress()` may be a better choice.

## Example:

Show the event object for the keyup handler (using a simple \$.print plugin) when a key is released in the input.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     fieldset { margin-bottom: 1em; }
6     input { display: block; margin-bottom: .25em;
7     #print-output {
8       width: 100%;
9     }
10    .print-output-line {
11      white-space: pre;
12      padding: 5px;
13      font-family: monaco, monospace;
14      font-size: .7em;
15    }
16
17  </style>
18  <script src="http://code.jquery.com/jquery-
19 </head>
20 <body>
21   <form>
22     <fieldset>
23       <label for="target">Type Something:</label>
24       <input id="target" type="text" />
25     </fieldset>
26   </form>
27   <button id="other">
28     Trigger the handler
29   </button>
```

```
30 <script type="text/javascript" src="/resources/events.js">
31 <script>
32 var xTriggered = 0;
33 $('#target').keyup(function(event) {
34     xTriggered++;
35     var msg = 'Handler for .keyup() called ' + xTriggered;
36     $.print(msg, 'html');
37     $.print(event);
38 }).keydown(function(event) {
39     if (event.which == 13) {
40         event.preventDefault();
41     }
42 });
43
44 $('#other').click(function() {
45     $('#target').keyup();
46 });</script>
47
48 </body>
49 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# :lang Selector

Categories: [Selectors](#) > [Basic Filter](#)

## lang selector

**Description:** *Selects all elements of the specified language.*

**jQuery( ":lang(language)" )**

version added: [1.9](#)

**language:** A language code.

The `:lang` selector matches elements that have a language value equal to the supplied language code or that start with the supplied language code immediately followed by "-". For example, the selector `$( "div:lang(en)" )` will match `<div lang="en">` and `<div lang="en-us">` (and any of their descendant `<div>`s), but not `<div lang="fr">`

For HTML elements, the language value is determined by the `lang` attribute and possibly information from `meta` elements or HTTP headers.

Further discussion of this usage can be found in the [W3C CSS specification](#).

## Example:

*Color div elements according to their language.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5  body { background-color: #ccc; }
6  h3 { margin: .25em 0; }
7  div { line-height: 1.5em}
8  .usa { background-color: #f00; color: #fff; }
9  .usa .usa { background-color: #fff; color: #00f; }
10 .usa .usa .usa { background-color: #00f; color: #fff; }
11
12 .spain { background-color: #f00; color: #ff0; }
13 .spain .spain { background-color: #ff0; color: #00f; }
14 .spain .spain .spain { background-color: #f00; color: #fff; }
15 </style>
16 <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
17 </head>
18 <body>
19
20 <h3>USA</h3>
21 <div lang="en-us">red
22   <div>white
23     <div>and blue</div>
24   </div>
25 </div>
26 <h3>España</h3>
27 <div lang="es-es">rojo
28   <div>amarillo
29     <div>y rojo</div>
30   </div>
31 </div>
```

```
32
33 <script>
34 $( "div:lang(en-us)" ).addClass( "usa" );
35 $( "div:lang(es-es)" ).addClass( "spain" );
36 </script>
37
38 </body>
39 </html>
```

Demo

POWERED BYHERONOTE

A new version of this book is available!

---

# .last()

Categories: [Traversing](#) > [Filtering](#)

`.last()`

Returns: [jQuery](#)

**Description:** Reduce the set of matched elements to the final one in the set.

`.last()`

version added: [1.4](#)

This method does not accept any arguments.

Given a jQuery object that represents a set of DOM elements, the `.last()` method constructs a new jQuery object from the last element in that set.

Consider a page with a simple list on it:

```
1 | <ul>
2 |   <li>list item 1</li>
3 |   <li>list item 2</li>
4 |   <li>list item 3</li>
5 |   <li>list item 4</li>
6 |   <li>list item 5</li>
7 | </ul>
```

We can apply this method to the set of list items:

```
1 | $('li').last().css('background-color', 'red');
```

The result of this call is a red background for the final item.

## Example:

*Highlight the last span in a paragraph.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>.highlight{background-color: yellow}.
5   <script src="http://code.jquery.com/jquery-
6 </head>
7 <body>
8   <p><span>Look:</span> <span>This is some te
9   <script>$("p span").last().addClass('highligh
10
11 </body>
12 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# :last-child Selector

Categories: [Selectors](#) > [Child Filter](#)

## last-child selector

**Description:** *Selects all elements that are the last child of their parent.*

**jQuery( ":last-child" )**

version added: [1.1.4](#)

While [:last](#) matches only a single element, `:last-child` can match more than one: one for each parent.

## Example:

*Find the last span in each matched div and add some css plus a hover state.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5   span.solast { text-decoration:line-through; }
6   </style>
7   <script src="http://code.jquery.com/jquery-
8 </head>
9 <body>
10  <div>
11    <span>John,</span>
12    <span>Karl,</span>
13    <span>Brandon,</span>
14    <span>Sam</span>
15  </div>
16  <div>
17    <span>Glen,</span>
18    <span>Tane,</span>
19
20    <span>Ralph,</span>
21    <span>David</span>
22  </div>
23 <script>
24 $( "div span:last-child" )
25   .css({color:"red", fontSize:"80%"})
26   .hover(function () {
27     $(this).addClass("solast");
28   }, function () {
29     $(this).removeClass("solast");
30   });
```

```
31 </script>
32
33 </body>
34 </html>
```

Demo

POWERED BY **HERONOTE**

A new version of this book is available!

---

# :last-of-type Selector

Categories: [Selectors](#) > [Child Filter](#)

## last-of-type selector

**Description:** *Selects all elements that are the last among siblings of the same element name.*

**jQuery( ":last-of-type" )**

version added: [1.9](#)

The `:last-of-type` selector matches elements that have no other element with the same parent and the same element name coming after it in the document tree.

## Example:

*Find the last span in each matched div and add some css plus a hover state.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5   span.solast { text-decoration:line-through; }
6   </style>
7   <script src="http://code.jquery.com/jquery-
8 </head>
9 <body>
10  <div>
11    <span>Corey, </span>
12    <span>Yehuda, </span>
13    <span>Adam, </span>
14    <span>Todd</span>
15  </div>
16  <div>
17    <span>Jörn, </span>
18    <span>Scott, </span>
19    <span>Timo, </span>
20    <b>Nobody</b>
21  </div>
22  <script>
23  $( "span:last-of-type" )
24    .css({color:"red", fontSize:"80%"})
25    .hover(function () {
26      $(this).addClass("solast");
27    }, function () {
28      $(this).removeClass("solast");
29    });
30  </script>
```

```
31  
32 </body>  
33 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# :last Selector

Categories: [Selectors](#) > [Basic Filter](#) | [Selectors](#) > [jQuery Extensions](#)

## last selector

**Description:** *Selects the last matched element.*

**jQuery( ":last" )**

version added: [1.0](#)

Note that `:last` selects a single element by filtering the current jQuery collection and matching the last element within it.

### Additional Notes:

Because `:last` is a jQuery extension and not part of the CSS specification, queries using `:last` cannot take advantage of the performance boost provided by the native DOM `querySelectorAll()` method. To achieve the best performance when using `:last` to select elements, first select the elements using a pure CSS selector, then use `.filter(":last")`.

## Example:

*Finds the last table row.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <script src="http://code.jquery.com/jquery-1.9.1.js"></script>
5  </head>
6  <body>
7    <table>
8
9    <tr><td>First Row</td></tr>
10   <tr><td>Middle Row</td></tr>
11   <tr><td>Last Row</td></tr>
12
13  </table>
14  <script>$("tr:last").css({backgroundColor: 'yellow'})
15
16  </body>
17  </html>
```

Demo

POWERED BY **HERONOTE**

A new version of this book is available!

---

# .length

Categories: [Properties](#) > [Properties of jQuery Object Instances](#)

**length**

Returns: [Number](#)

**Description:** *The number of elements in the jQuery object.*

**length**

version added: [1.0](#)

The number of elements currently matched. The [.size\(\)](#) method will return the same value.

## Example:

Count the divs. Click to add more.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     body { cursor:pointer; }
6     div { width:50px; height:30px; margin:5px;
7           background:green; }
8     span { color:red; }
9   </style>
10  <script src="http://code.jquery.com/jquery-
11 </head>
12 <body>
13   <span></span>
14   <div></div>
15  <script>$(document.body).click(function () {
16    $(document.body).append("<div>");
17    var n = $("div").length;
18    $("span").text("There are " + n + " divs."
19                  "Click to add more.");
20  }).trigger('click'); // trigger the click to
21 </script>
22
23 </body>
24 </html>
```

Demo

**A new version of this book is available!**

---

# .live()

Categories: [Events](#) > [Event Handler Attachment](#)

**.live( events, handler(eventObject) )** *Returns:* [jQuery](#)  
version deprecated: [1.7](#), removed: [1.9](#)

**Description:** *Attach an event handler for all elements which match the current selector, now and in the future.*

**.live( events, handler(eventObject) )** version added: [1.3](#)

**events**

Type: [String](#)

A string containing a JavaScript event type, such as "click" or "keydown." As of jQuery 1.4 the string can contain multiple, space-separated event types or custom event names.

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute at the time the event is triggered.

**.live( events, data, handler(eventObject) )** version added: [1.4](#)

**events**

Type: [String](#)

A string containing a JavaScript event type, such as "click" or "keydown." As of jQuery 1.4 the string can contain multiple, space-separated event types or custom event names.

**data**

Type: [PlainObject](#)

An object containing data that will be passed to the event handler.

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute at the time the event is triggered.

**.live( events )** version added: [1.4.3](#)

**events**

Type: [PlainObject](#)

A plain object of one or more JavaScript event types and functions to execute for them.

As of jQuery 1.7, the `.live()` method is deprecated. Use `.on()` to attach event handlers. Users of older versions of jQuery should use `.delegate()` in preference to `.live()`.

This method provides a means to attach delegated event handlers to the `document` element of a page, which simplifies the use of event handlers when content is dynamically added to a page. See the discussion of direct versus delegated events in the `.on()` method for more information.

Rewriting the `.live()` method in terms of its successors is straightforward; these are templates for equivalent calls for all three event attachment methods:

```
1 | $(selector).live(events, data, handler);  
2 | $(document).delegate(selector, events, data, handler);  
3 | $(document).on(events, selector, data, handler);
```

The `events` argument can either be a space-separated list of event type names and optional namespaces, or an object of event name strings and handlers. The `data` argument is optional and can be omitted. For example, the following three method calls are functionally equivalent (but see below for more effective and performant ways to attach delegated event handlers):

```
1 | $("a.offsite").live("click", function(){ alert("offsite"); });  
2 | $(document).delegate("a.offsite", "click", function(){ alert("offsite"); });  
3 | $(document).on("click", "a.offsite", function(){ alert("offsite"); });
```

Use of the `.live()` method is no longer recommended since later versions of jQuery offer better methods that do not have its drawbacks. In particular, the following issues arise with the use of `.live()`:

jQuery attempts to retrieve the elements specified by the selector before calling the `.live()` method, which may be time-consuming on large documents.

Chaining methods is not supported. For example, `$("#a").find(".offsite, .external").live( ... );` is *not* valid and does not work as expected.

Since all `.live()` events are attached at the `document` element, events take the longest and slowest possible path before they are handled.

On mobile iOS (iPhone, iPad and iPod Touch) the `click` event does not bubble to the document body for most elements and cannot be used with `.live()` without applying one of the following workarounds:

- 1 Use natively clickable elements such as `a` or `button`, as both of these do bubble to `document`.
- 2 Use `.on()` or `.delegate()` attached to an element below the level of `document.body`, since mobile iOS does bubble within the body.
- 3 Apply the CSS style `cursor:pointer` to the element that needs to bubble clicks (or a parent including `document.documentElement`). Note however, this will disable copy/paste on the element and cause it to be highlighted when touched.

Calling `event.stopPropagation()` in the event handler is ineffective in stopping event handlers attached lower in the document; the event has already propagated to `document`.

The `.live()` method interacts with other event methods in ways that can be surprising, e.g., `$(document).off("click")` removes all click handlers attached by any call to `.live()`!

For pages still using `.live()`, this list of version-specific differences may be helpful:

Before jQuery 1.7, to stop further handlers from executing after one bound using `.live()`, the handler must return `false`. Calling `.stopPropagation()` will not accomplish this.

As of **jQuery 1.4** the `.live()` method supports custom events

as well as *all JavaScript events that bubble*. It also supports certain events that don't bubble, including `change`, `submit`, `focus` and `blur`.

In **jQuery 1.3.x** only the following JavaScript events could be bound: `click`, `dblclick`, `keydown`, `keypress`, `keyup`, `mousedown`, `mousemove`, `mouseout`, `mouseover`, and `mouseup`.

## Examples:

**Example:** *Cancel a default action and prevent it from bubbling up by returning false.*

```
1 | $("a").live("click", function() { return false
```

**Example:** *Cancel only the default action by using the preventDefault method.*

```
1 | $("a").live("click", function(event){
2 |     event.preventDefault();
3 | });
```

**Example:** *Bind custom events with .live().*

```
1 | $("p").live("myCustomEvent", function(e, myName)
2 |     $("this").text("Hi there!");
3 |     $("span").stop().css("opacity", 1)
4 |         .text("myName = " + myName)
5 |         .fadeIn(30).fadeOut(1000);
6 | });
7 | $("button").click(function () {
8 |     $("p").trigger("myCustomEvent");
9 | });
```

**Example:** *Use an object to bind multiple live event handlers. Note that live() calls the click mouseover and mouseout event handlers for all paragraphs--even new ones.*

```
1  $("p").live({
2    click: function() {
3      $(this).after("<p>Another paragraph!</p>");
4    },
5    mouseover: function() {
6      $(this).addClass("over");
7    },
8    mouseout: function() {
9      $(this).removeClass("over");
10   }
11 });
```

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# .load()

Categories: [Ajax](#) > [Shorthand Methods](#)

```
.load( url [, data ] [, complete(responseText, textStatus, XMLHttpRequest) ] ) Returns: jQuery
```

**Description:** *Load data from the server and place the returned HTML into the matched element.*

```
.load( url [, data ] [, version added: 1.0  
complete(responseText, textStatus, XMLHttpRequest)  
] )
```

**url**

Type: [String](#)

A string containing the URL to which the request is sent.

**data**

Type: [PlainObject](#) or [String](#)

A plain object or string that is sent to the server with the request.

**complete(responseText, textStatus, XMLHttpRequest)**

Type: [Function\(\)](#)

A callback function that is executed when the request completes.

*Note: The event handling suite also has a method named [.load\(\)](#). jQuery determines which method to fire based on the set of arguments passed to it.*

This method is the simplest way to fetch data from the server. It is roughly equivalent to `$.get(url, data, success)` except that it is a method rather than global function and it has an implicit callback function. When a successful response is detected (i.e. when `textStatus` is "success" or "notmodified"), `.load()` sets the HTML

contents of the matched element to the returned data. This means that most uses of the method can be quite simple:

```
1 | $('#result').load('ajax/test.html');
```

If no element is matched by the selector — in this case, if the document does not contain an element with `id="result"` — the Ajax request will *not* be sent.

### Callback Function

If a "complete" callback is provided, it is executed after post-processing and HTML insertion has been performed. The callback is fired once for each element in the jQuery collection, and `this` is set to each DOM element in turn.

```
1 | $('#result').load('ajax/test.html', function(  
2 |     alert('Load was performed.');
```

```
3 | });
```

In the two examples above, if the current document does not contain an element with an ID of "result," the `.load()` method is not executed.

### Request Method

The POST method is used if data is provided as an object; otherwise, GET is assumed.

### Loading Page Fragments

The `.load()` method, unlike `$.get()`, allows us to specify a portion of the remote document to be inserted. This is achieved with a special syntax for the `url` parameter. If one or more space characters are included in the string, the portion of the string following the first space is assumed to be a jQuery selector that determines the content to be loaded.

We could modify the example above to use only part of the document that is fetched:

```
1 | $('#result').load('ajax/test.html #container')
```

When this method executes, it retrieves the content of `ajax/test.html`, but then jQuery parses the returned document to find the element with an ID of `container`. This element, along with its contents, is inserted into the element with an ID of `result`, and the rest of the retrieved document is discarded.

jQuery uses the browser's `.innerHTML` property to parse the retrieved document and insert it into the current document. During this process, browsers often filter elements from the document such as `<html>`, `<title>`, or `<head>` elements. As a result, the elements retrieved by `.load()` may not be exactly the same as if the document were retrieved directly by the browser.

### Script Execution

When calling `.load()` using a URL without a suffixed selector expression, the content is passed to `.html()` prior to scripts being removed. This executes the script blocks before they are discarded. If `.load()` is called with a selector expression appended to the URL, however, the scripts are stripped out prior to the DOM being updated, and thus are *not* executed. An example of both cases can be seen below:

Here, any JavaScript loaded into `#a` as a part of the document will successfully execute.

```
1 | $('#a').load('article.html');
```

However, in the following case, script blocks in the document being loaded into `#b` are stripped out and not executed:

```
1 | $('#b').load('article.html #target');
```

## Additional Notes:

Due to browser security restrictions, most "Ajax" requests are subject to the [same origin policy](#); the request can not successfully retrieve data from a different domain, subdomain, or protocol.

## Examples:

**Example:** Load the main page's footer navigation into an ordered list.

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5    body{ font-size: 12px; font-family: Arial; }
6    </style>
7    <script src="http://code.jquery.com/jquery-...
8  </head>
9  <body>
10
11  <b>Footer navigation:</b>
12  <ol id="new-nav"></ol>
13
14  <script>
15    $("#new-nav").load("/ #jq-footerNavigation .
16  </script>
17
18  </body>
19  </html>
```

Demo

**Example:** Display a notice if the Ajax request encounters an error.

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
```

```

5   body{ font-size: 12px; font-family: Arial; }
6   </style>
7   <script src="http://code.jquery.com/jquery-latest.js"
8 </head>
9 <body>
10
11 <b>Successful Response (should be blank):</b>
12 <div id="success"></div>
13 <b>Error Response:</b>
14 <div id="error"></div>
15
16 <script>
17 $( "#success" ).load( "/not-here.php", function( response, s
18   if ( status == "error" ) {
19     var msg = "Sorry but there was an error: ";
20     $( "#error" ).html( msg + xhr.status + " " + xhr.status
21   }
22 });
23 </script>
24
25 </body>
26 </html>

```

## Demo

**Example:** I load the feeds.html file into the div with the ID of feeds.

```
1 | $( "#feeds" ).load( "feeds.html" );
```

## Result:

```
1 | <div id="feeds"><b>45</b> feeds found.</div>
```

**Example:** *pass arrays of data to the server.*

```
1 | $("#objectID").load("test.php", { 'choices[]':
```

**Example:** *Same as above but will POST the additional parameters to the server and a callback that is executed when the server is finished responding.*

```
1 | $("#feeds").load("feeds.php", {limit: 25}, func  
2 | alert("The last 25 entries in the feed have be  
3 | });
```

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# .load()

Categories: [Events](#) > [Document Loading](#)

`.load( handler(eventObject) )`

Returns: [jQuery](#)  
version deprecated: [1.8](#)

**Description:** *Bind an event handler to the "load" JavaScript event.*

`.load( handler(eventObject) )`

version added: [1.0](#)

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute when the event is triggered.

`.load( [eventData ],  
handler(eventObject) )`

version added: [1.4.3](#)

**eventData**

Type: [PlainObject](#)

An object containing data that will be passed to the event handler.

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute each time the event is triggered.

This method is a shortcut for `.on('load', handler)`.

The `load` event is sent to an element when it and all sub-elements have been completely loaded. This event can be sent to any element associated with a URL: images, scripts, frames, iframes, and the `window` object.

For example, consider a page with a simple image:

```
1 | 
```

The event handler can be bound to the image:

```
1 | $( '#book' ).load(function() {  
2 |   // Handler for .load() called.  
3 | });
```

As soon as the image has been loaded, the handler is called.

In general, it is not necessary to wait for all images to be fully loaded. If code can be executed earlier, it is usually best to place it in a handler sent to the `.ready()` method.

*The Ajax module also has a method named `.load()`. Which one is fired depends on the set of arguments passed.*

### **Caveats of the `load` event when used with images**

*A common challenge developers attempt to solve using the `.load()` shortcut is to execute a function when an image (or collection of images) have completely loaded. There are several known caveats with this that should be noted. These are:*

It doesn't work consistently nor reliably cross-browser

It doesn't fire correctly in WebKit if the image src is set to the same src as before

It doesn't correctly bubble up the DOM tree

Can cease to fire for images that already live in the browser's cache

**Note:** The `.live()` and `.delegate()` methods cannot be used to detect the `load` event of an `iframe`. The `load` event does not correctly bubble up the parent document and the `event.target` isn't set by Firefox, IE9 or Chrome, which is required to do event delegation.

## Examples:

**Example:** *Run a function when the page is fully loaded including graphics.*

```
1 | $(window).load(function () {  
2 |     // run code  
3 | });
```

**Example:** *Add the class `bigImg` to all images with height greater than 100 upon each image load.*

```
1 | $('img.userIcon').load(function(){  
2 |     if($(this).height() > 100) {  
3 |         $(this).addClass('bigImg');  
4 |     }  
5 | });
```

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# :lt() Selector

Categories: [Selectors](#) > [Basic Filter](#) | [Selectors](#) > [jQuery Extensions](#)

## lt selector

**Description:** *Select all elements at an index less than `index` within the matched set.*

**jQuery( ":lt(index)" )**

version added: [1.0](#)

**index:** Zero-based index.

### index-related selectors

The index-related selectors (including this "less than" selector) filter the set of elements that have matched the expressions that precede them. They narrow the set down based on the order of the elements within this matched set. For example, if elements are first selected with a class selector ( `.myclass` ) and four elements are returned, these elements are given indices 0 through 3 for the purposes of these selectors.

Note that since JavaScript arrays use *0-based indexing*, these selectors reflect that fact. This is why `$('.myclass:lt(1)')` selects the first element in the document with the class `myclass`, rather than selecting no elements. In contrast, `:nth-child(n)` uses *1-based indexing* to conform to the CSS specification.

### Additional Notes:

Because `:lt()` is a jQuery extension and not part of the CSS specification, queries using `:lt()` cannot take advantage of the performance boost provided by the native DOM `querySelectorAll()` method. For better performance in modern browsers, use `$("#your-pure-css-selector").slice(0, index)` instead.

## Example:

*Find TDs less than the one with the 4th index (TD#4).*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <script src="http://code.jquery.com/jquery-1.10.2.min.js"></script>
5  </head>
6  <body>
7    <table border="1">
8
9    <tr><td>TD #0</td><td>TD #1</td><td>TD #2</td><td>TD #3</td><td>TD #4</td><td>TD #5</td></tr>
10   <tr><td>TD #3</td><td>TD #4</td><td>TD #5</td><td>TD #6</td><td>TD #7</td><td>TD #8</td></tr>
11
12   <tr><td>TD #6</td><td>TD #7</td><td>TD #8</td><td>TD #9</td><td>TD #10</td><td>TD #11</td></tr>
13 </table>
14 <script>$("td:lt(4)").css("color", "red");</script>
15
16 </body>
17 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# .map()

Categories: [Traversing](#) > [Filtering](#)

`.map( callback(index, domElement) )`

Returns: [jQuery](#)

**Description:** *Pass each element in the current matched set through a function, producing a new jQuery object containing the return values.*

`.map( callback(index, domElement) )` version added: [1.2](#)

**callback(index, domElement)**

Type: [Function\(\)](#)

A function object that will be invoked for each element in the current set.

If you wish to process a plain array or object, use the [jQuery.map\(\)](#) instead.

As the return value is a jQuery object, which contains an array, it's very common to call `.get()` on the result to work with a basic array.

The `.map()` method is particularly useful for getting or setting the value of a collection of elements. Consider a form with a set of checkboxes in it:

```
1 <form method="post" action="">
2   <fieldset>
3     <div>
4       <label for="two">2</label>
5       <input type="checkbox" value="2" id="t
6     </div>
7     <div>
8       <label for="four">4</label>
9       <input type="checkbox" value="4" id="f
10    </div>
11    <div>
12      <label for="six">6</label>
13      <input type="checkbox" value="6" id="s:
14    </div>
```

```
15 |     <div>
16 |         <label for="eight">8</label>
17 |         <input type="checkbox" value="8" id="eight"
18 |     </div>
19 | </fieldset>
20 | </form>
```

To get a comma-separated list of checkbox `IDs`:

```
1 | $(':checkbox').map(function() {
2 |     return this.id;
3 | }).get().join();
```

The result of this call is the string, `"two, four, six, eight"`.

Within the callback function, `this` refers to the current DOM element for each iteration. The function can return an individual data item or an array of data items to be inserted into the resulting set. If an array is returned, the elements inside the array are inserted into the set. If the function returns `null` or `undefined`, no element will be inserted.

## Examples:

**Example:** *Build a list of all the values within a form.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5      p { color:red; }
6    </style>
7    <script src="http://code.jquery.com/jquery-1.9.1.js"></script>
8  </head>
9  <body>
10   <p><b>Values: </b></p>
11   <form>
12     <input type="text" name="name" value="John">
13
14     <input type="text" name="password" value="123456">
15     <input type="text" name="url" value="http://www.google.com">
16
17   </form>
18   <script>
19     $("p").append( $("input").map(function(){
20       return $(this).val();
21     }).get().join(", ") );
22   </script>
23
24 </body>
25 </html>
```

Demo

**Example:** *A contrived example to show some functionality.*

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5    body { font-size:16px; }
6    ul { float:left; margin:0 30px; color:blue; }
7    #results { color:red; }
8    </style>
9    <script src="http://code.jquery.com/jquery-latest.js">
10 </head>
11 <body>
12   <ul>
13     <li>First</li>
14     <li>Second</li>
15     <li>Third</li>
16
17     <li>Fourth</li>
18     <li>Fifth</li>
19   </ul>
20   <ul id="results">
21
22   </ul>
23 <script>
24 var mappedItems = $("li").map(function (index) {
25   var replacement = $("<li>").text($(this).text()).get(
26   if (index == 0) {
27     /* make the first item all caps */
28     $(replacement).text($(replacement).text().toUpperCase
29   } else if (index == 1 || index == 3) {
30     /* delete the second and fourth items */
31     replacement = null;
32   } else if (index == 2) {
33     /* make two of the third item and add some text */
34     replacement = [replacement,$("<li>").get(0)];
35     $(replacement[0]).append("<b> - A</b>");
36     $(replacement[1]).append("Extra <b> - B</b>")

```

```

37     }
38
39     /* replacement will be a dom element, null,
40        or an array of dom elements */
41     return replacement;
42 });
43 $("#results").append(mappedItems);
44
45 </script>
46
47 </body>
48 </html>

```

Demo

**Example:** *Equalize the heights of the divs.*

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5    div { width: 40px; float:left; }
6    input { clear:left}
7    </style>
8    <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
9  </head>
10 <body>
11
12
13 <input type="button" value="equalize div heights" />
14
15 <div style="background:red; height: 40px; width: 40px; float:left; margin-right: 10px;"></div>
16 <div style="background:green; height: 70px; width: 40px; float:left; margin-right: 10px;"></div>
17 <div style="background:blue; height: 50px; width: 40px; float:left; margin-right: 10px;"></div>
18
19

```

```
20 <script>
21 $.fn.equalizeHeights = function() {
22     var maxHeight = this.map(function(i,e) {
23         return $(e).height();
24     }).get();
25
26     return this.height( Math.max.apply(this, maxHeight) );
27 };
28
29 $('input').click(function(){
30     $('div').equalizeHeights();
31 });
32
33 </script>
34
35 </body>
36 </html>
```

Demo

POWERED BY HERONOTE

A new version of this book is available!

---

# .mousedown()

Categories: [Events](#) > [Mouse Events](#)

`.mousedown( handler(eventObject) )`

Returns: [jQuery](#)

**Description:** Bind an event handler to the "mousedown" JavaScript event, or trigger that event on an element.

`.mousedown( handler(eventObject) )` version added: [1.0](#)

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute each time the event is triggered.

`.mousedown( [eventData ], handler(eventObject) )` version added: [1.4.3](#)

**eventData**

Type: [PlainObject](#)

An object containing data that will be passed to the event handler.

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute each time the event is triggered.

`.mousedown()` version added: [1.0](#)

This method does not accept any arguments.

This method is a shortcut for `.on('mousedown', handler)` in the first variation, and `.trigger('mousedown')` in the second.

The `mousedown` event is sent to an element when the mouse pointer is over the element, and the mouse button is pressed. Any HTML element can receive this event.

For example, consider the HTML:

```
1 | <div id="target">
2 |   Click here
3 | </div>
```

```
4 | <div id="other">
5 |   Trigger the handler
6 | </div>
```

Click here

Trigger the handler

The event handler can be bound to any `<div>`:

```
1 | $('#target').mousedown(function() {
2 |   alert('Handler for .mousedown() called.');
```

```
3 | });
```

Now if we click on this element, the alert is displayed:

Handler for `.mousedown()` called.

We can also trigger the event when a different element is clicked:

```
1 | $('#other').click(function() {
2 |   $('#target').mousedown();
3 | });
```

After this code executes, clicks on Trigger the handler will also alert the message.

The `mousedown` event is sent when any mouse button is clicked. To act only on specific buttons, we can use the event object's `which` property. Not all browsers support this property (Internet Explorer uses `button` instead), but jQuery normalizes the property so that it is safe to use in any browser. The value of `which` will be 1 for the left button, 2 for the middle button, or 3 for the right button.

This event is primarily useful for ensuring that the primary button

was used to begin a drag operation; if ignored, strange results can occur when the user attempts to use a context menu. While the middle and right buttons can be detected with these properties, this is not reliable. In Opera and Safari, for example, right mouse button clicks are not detectable by default.

If the user clicks on an element, drags away from it, and releases the button, this is still counted as a `mousedown` event. This sequence of actions is treated as a "canceling" of the button press in most user interfaces, so it is usually better to use the `click` event unless we know that the `mousedown` event is preferable for a particular situation.

## Example:

*Show texts when mouseup and mousedown event triggering.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <script src="http://code.jquery.com/jquery-1.11.0.min.js"></script>
5  </head>
6  <body>
7    <p>Press mouse and release here.</p>
8
9    <script>
10     $( "p" ).mouseup( function() {
11       $( this ).append( '<span style="color:#F00">Mouse up!</span>' );
12     } ).mousedown( function() {
13       $( this ).append( '<span style="color:#00F">Mouse down!</span>' );
14     } );
15
16   </script>
17
18 </body>
19 </html>
```

Demo

**A new version of this book is available!**

---

# .mouseenter()

Categories: [Events](#) > [Mouse Events](#)

`.mouseenter( handler(eventObject) )`

Returns: [jQuery](#)

**Description:** *Bind an event handler to be fired when the mouse enters an element, or trigger that handler on an element.*

`.mouseenter( handler(eventObject) )` version added: [1.0](#)

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute each time the event is triggered.

`.mouseenter( [eventData ], handler(eventObject) )` version added: [1.4.3](#)

**eventData**

Type: [PlainObject](#)

An object containing data that will be passed to the event handler.

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute each time the event is triggered.

`.mouseenter()` version added: [1.0](#)

This method does not accept any arguments.

This method is a shortcut for `.on('mouseenter', handler)` in the first two variations, and `.trigger('mouseenter')` in the third.

The `mouseenter` JavaScript event is proprietary to Internet Explorer. Because of the event's general utility, jQuery simulates this event so that it can be used regardless of browser. This event is sent to an element when the mouse pointer enters the element. Any HTML element can receive this event.

For example, consider the HTML:

```

1 | <div id="outer">
2 |   Outer
3 |   <div id="inner">
4 |     Inner
5 |   </div>
6 | </div>
7 | <div id="other">
8 |   Trigger the handler
9 | </div>
10| <div id="log"></div>

```



The event handler can be bound to any element:

```

1 | $('#outer').mouseenter(function() {
2 |     $('#log').append('<div>Handler for .mouseenter');
3 | });

```

Now when the mouse pointer moves over the Outer `<div>`, the message is appended to `<div id="log">`. You can also trigger the event when another element is clicked:

```

1 | $('#other').click(function() {
2 |     $('#outer').mouseenter();
3 | });

```

After this code executes, clicks on Trigger the handler will also

append the message.

The `mouseenter` event differs from `mouseover` in the way it handles event bubbling. If `mouseover` were used in this example, then when the mouse pointer moved over the Inner element, the handler would be triggered. This is usually undesirable behavior. The `mouseenter` event, on the other hand, only triggers its handler when the mouse enters the element it is bound to, not a descendant. So in this example, the handler is triggered when the mouse enters the Outer element, but not the Inner element.

---

## Example:

Show texts when `mouseenter` and `mouseout` event triggering. `mouseover` fires when the pointer moves into the child element as well. while `mouseenter` fires only when the pointer moves into the bound element.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     div.out {
6       width:40%;
7       height:120px;
8       margin:0 15px;
9       background-color:#D6EDFC;
10      float:left;
11    }
12    div.in {
13      width:60%;
14      height:60%;
15      background-color:#FFCC00;
16      margin:10px auto;
17    }
18    p {
19      line-height:1em;
20      margin:0;
21      padding:0;
22    }
23  </style>
24  <script src="http://code.jquery.com/jquery-
25  </head>
26  <body>
27
28  <div class="out overout"><p>move your mouse</
```

```
29
30 <div class="out enterleave"><p>move your mouse</
31
32
33 <script>
34     var i = 0;
35     $("div.overout").mouseover(function(){
36         $("p:first",this).text("mouse over");
37         $("p:last",this).text(++i);
38     }).mouseout(function(){
39         $("p:first",this).text("mouse out");
40     });
41
42     var n = 0;
43     $("div.enterleave").mouseenter(function(){
44         $("p:first",this).text("mouse enter");
45         $("p:last",this).text(++n);
46     }).mouseleave(function(){
47         $("p:first",this).text("mouse leave");
48     });
49
50 </script>
51
52 </body>
53 </html>
```

Demo

**A new version of this book is available!**

---

# .mouseleave()

Categories: [Events](#) > [Mouse Events](#)

`.mouseleave( handler(eventObject) )`

Returns: [jQuery](#)

**Description:** *Bind an event handler to be fired when the mouse leaves an element, or trigger that handler on an element.*

`.mouseleave( handler(eventObject) )` version added: [1.0](#)

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute each time the event is triggered.

`.mouseleave( [ eventData ], handler(eventObject) )` version added: [1.4.3](#)

**eventData**

Type: [PlainObject](#)

An object containing data that will be passed to the event handler.

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute each time the event is triggered.

`.mouseleave()` version added: [1.0](#)

This method does not accept any arguments.

This method is a shortcut for `.on('mouseleave', handler)` in the first two variations, and `.trigger('mouseleave')` in the third.

The `mouseleave` JavaScript event is proprietary to Internet Explorer. Because of the event's general utility, jQuery simulates this event so that it can be used regardless of browser. This event is sent to an element when the mouse pointer leaves the element. Any HTML element can receive this event.

For example, consider the HTML:

```

1 | <div id="outer">
2 |   Outer
3 |   <div id="inner">
4 |     Inner
5 |   </div>
6 | </div>
7 | <div id="other">
8 |   Trigger the handler
9 | </div>
10| <div id="log"></div>

```



The event handler can be bound to any element:

```

1 | $('#outer').mouseleave(function() {
2 |   $('#log').append('<div>Handler for .mouseleave');
3 | });

```

Now when the mouse pointer moves out of the Outer `<div>`, the message is appended to `<div id="log">`. You can also trigger the event when another element is clicked:

```

1 | $('#other').click(function() {
2 |   $('#outer').mouseleave();
3 | });

```

After this code executes, clicks on Trigger the handler will also

append the message.

The `mouseleave` event differs from `mouseout` in the way it handles event bubbling. If `mouseout` were used in this example, then when the mouse pointer moved out of the Inner element, the handler would be triggered. This is usually undesirable behavior. The `mouseleave` event, on the other hand, only triggers its handler when the mouse leaves the element it is bound to, not a descendant. So in this example, the handler is triggered when the mouse leaves the Outer element, but not the Inner element.

---

## Example:

Show number of times `mouseout` and `mouseleave` events are triggered. `mouseout` fires when the pointer moves out of child element as well while `mouseleave` fires only when the pointer moves out of the bound element.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5   div.out {
6     width:40%;
7     height:120px;
8     margin:0 15px;
9     background-color:#D6EDFC;
10    float:left;
11  }
12  div.in {
13    width:60%;
14    height:60%;
15    background-color:#FFCC00;
16    margin:10px auto;
17  }
18  p {
19    line-height:1em;
20    margin:0;
21    padding:0;
22  }
23 </style>
24   <script src="http://code.jquery.com/jquery-
25 </head>
26 <body>
27
```

```
28 <div class="out overout"><p>move your mouse</p>
29
30 <div class="out enterleave"><p>move your mouse</
31
32
33 <script>
34     var i = 0;
35     $("div.overout").mouseover(function(){
36         $("p:first",this).text("mouse over");
37     }).mouseout(function(){
38         $("p:first",this).text("mouse out");
39         $("p:last",this).text(++i);
40     });
41
42     var n = 0;
43     $("div.enterleave").mouseenter(function(){
44         $("p:first",this).text("mouse enter");
45     }).mouseleave(function(){
46         $("p:first",this).text("mouse leave");
47         $("p:last",this).text(++n);
48     });
49
50 </script>
51
52 </body>
53 </html>
```

Demo

**A new version of this book is available!**

---

# .mousemove()

Categories: [Events](#) > [Mouse Events](#)

`.mousemove( handler(eventObject) )`

Returns: [jQuery](#)

**Description:** *Bind an event handler to the "mousemove" JavaScript event, or trigger that event on an element.*

`.mousemove( handler(eventObject) )` version added: [1.0](#)

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute each time the event is triggered.

`.mousemove( [eventData ], handler(eventObject) )` version added: [1.4.3](#)

**eventData**

Type: [PlainObject](#)

An object containing data that will be passed to the event handler.

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute each time the event is triggered.

`.mousemove()` version added: [1.0](#)

This method does not accept any arguments.

This method is a shortcut for `.on('mousemove', handler)` in the first two variations, and `.trigger('mousemove')` in the third.

The `mousemove` event is sent to an element when the mouse pointer moves inside the element. Any HTML element can receive this event.

For example, consider the HTML:

```
1 | <div id="target">
2 |   Move here
3 | </div>
```

```
4 | <div id="other">
5 |     Trigger the handler
6 | </div>
7 | <div id="log"></div>
```

The event handler can be bound to the target:

```
1 | $("#target").mousemove(function(event) {
2 |     var msg = "Handler for .mousemove() called at ";
3 |     msg += event.pageX + ", " + event.pageY;
4 |     $("#log").append("<div>" + msg + "</div>");
5 | });
```

Now when the mouse pointer moves within the target button, the messages are appended to <div id="log">:

Handler for .mousemove() called at (399, 48) Handler for  
.mousemove() called at (398, 46)  
Handler for .mousemove() called at (397, 44)  
Handler for .mousemove() called at (396, 42)

To trigger the event manually, apply `.mousemove()` without an argument:

```
1 | $("#other").click(function() {
2 |     $("#target").mousemove();
3 | });
```

After this code executes, clicks on the Trigger button will also append the message:

Handler for .mousemove() called at (undefined, undefined)

When tracking mouse movement, you usually need to know the actual position of the mouse pointer. The event object that is passed to the handler contains some information about the mouse

coordinates. Properties such as `.clientX`, `.offsetX`, and `.pageX` are available, but support for them differs between browsers.

Fortunately, jQuery normalizes the `.pageX` and `.pageY` properties so that they can be used in all browsers. These properties provide the X and Y coordinates of the mouse pointer relative to the top-left corner of the document, as illustrated in the example output above.

Keep in mind that the `mousemove` event is triggered whenever the mouse pointer moves, even for a pixel. This means that hundreds of events can be generated over a very small amount of time. If the handler has to do any significant processing, or if multiple handlers for the event exist, this can be a serious performance drain on the browser. It is important, therefore, to optimize `mousemove` handlers as much as possible, and to unbind them as soon as they are no longer needed.

A common pattern is to bind the `mousemove` handler from within a `mousedown` handler, and to unbind it from a corresponding `mouseup` handler. If implementing this sequence of events, remember that the `mouseup` event might be sent to a different HTML element than the `mousemove` event was. To account for this, the `mouseup` handler should typically be bound to an element high up in the DOM tree, such as `<body>`.



```
30  
31 </body>  
32 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# .mouseout()

Categories: [Events](#) > [Mouse Events](#)

`.mouseout( handler(eventObject) )`

Returns: [jQuery](#)

**Description:** Bind an event handler to the "mouseout" JavaScript event, or trigger that event on an element.

`.mouseout( handler(eventObject) )`

version added: [1.0](#)

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute each time the event is triggered.

`.mouseout( [eventData ],  
handler(eventObject) )`

version added: [1.4.3](#)

**eventData**

Type: [PlainObject](#)

An object containing data that will be passed to the event handler.

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute each time the event is triggered.

`.mouseout()`

version added: [1.0](#)

This method does not accept any arguments.

This method is a shortcut for `.on('mouseout', handler)` in the first two variations, and `.trigger('mouseout')` in the third.

The `mouseout` event is sent to an element when the mouse pointer leaves the element. Any HTML element can receive this event.

For example, consider the HTML:

```
1 | <div id="outer">  
2 |   Outer  
3 |   <div id="inner">
```

```
4     Inner
5     </div>
6 </div>
7 <div id="other">
8     Trigger the handler
9 </div>
10 <div id="log"></div>
```



The event handler can be bound to any element:

```
1 | $( '#outer' ).mouseout( function() {
2 |     $( '#log' ).append( 'Handler for .mouseout() ca
3 | });
```

Now when the mouse pointer moves out of the Outer `<div>`, the message is appended to `<div id="log">`. To trigger the event manually, apply `.mouseout()` without an argument::

```
1 | $( '#other' ).click( function() {
2 |     $( '#outer' ).mouseout();
3 | });
```

After this code executes, clicks on Trigger the handler will also append the message.

This event type can cause many headaches due to event bubbling.

For instance, when the mouse pointer moves out of the Inner element in this example, a `mouseout` event will be sent to that, then trickle up to Outer. This can trigger the bound `mouseout` handler at inopportune times. See the discussion for `.mouseleave()` for a useful alternative.

## Example:

Show the number of times `mouseout` and `mouseleave` events are triggered. `mouseout` fires when the pointer moves out of the child element as well while `mouseleave` fires only when the pointer moves out of the bound element.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5   div.out {
6     width:40%;
7     height:120px;
8     margin:0 15px;
9     background-color:#D6EDFC;
10    float:left;
11  }
12  div.in {
13    width:60%;
14    height:60%;
15    background-color:#FFCC00;
16    margin:10px auto;
17  }
18  p {
19    line-height:1em;
20    margin:0;
21    padding:0;
22  }
23 </style>
24   <script src="http://code.jquery.com/jquery-
25 </head>
26 <body>
27
```

```
28 <div class="out overout"><p>move your mouse</p>
29
30 <div class="out enterleave"><p>move your mouse</
31
32
33 <script>
34 var i = 0;
35 $("div.overout").mouseout(function(){
36   $("p:first", this).text("mouse out");
37   $("p:last", this).text(++i);
38 }).mouseover(function(){
39   $("p:first", this).text("mouse over");
40 });
41
42 var n = 0;
43 $("div.enterleave").on("mouseenter", function(){
44   $("p:first", this).text("mouse enter");
45 }).on("mouseleave", function(){
46   $("p:first", this).text("mouse leave");
47   $("p:last", this).text(++n);
48 });
49
50 </script>
51
52 </body>
53 </html>
```

Demo

**A new version of this book is available!**

---

# .mouseover()

Categories: [Events](#) > [Mouse Events](#)

`.mouseover( handler(eventObject) )`

Returns: [jQuery](#)

**Description:** *Bind an event handler to the "mouseover" JavaScript event, or trigger that event on an element.*

`.mouseover( handler(eventObject) )`

version added: [1.0](#)

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute each time the event is triggered.

`.mouseover( [eventData ],  
handler(eventObject) )`

version added: [1.4.3](#)

**eventData**

Type: [PlainObject](#)

An object containing data that will be passed to the event handler.

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute each time the event is triggered.

`.mouseover()`

version added: [1.0](#)

This method does not accept any arguments.

This method is a shortcut for `.on('mouseover', handler)` in the first two variations, and `.trigger('mouseover')` in the third.

The `mouseover` event is sent to an element when the mouse pointer enters the element. Any HTML element can receive this event.

For example, consider the HTML:

```
1 | <div id="outer">
2 |   Outer
3 |   <div id="inner">
```

```
4     Inner
5     </div>
6 </div>
7 <div id="other">
8     Trigger the handler
9 </div>
10 <div id="log"></div>
```



The event handler can be bound to any element:

```
1 | $( '#outer' ).mouseover( function() {
2 |     $( '#log' ).append( '<div>Handler for .mouseover'
3 | });
```

Now when the mouse pointer moves over the Outer `<div>`, the message is appended to `<div id="log">`. We can also trigger the event when another element is clicked:

```
1 | $( '#other' ).click( function() {
2 |     $( '#outer' ).mouseover();
3 | });
```

After this code executes, clicks on Trigger the handler will also append the message.

This event type can cause many headaches due to event bubbling.

For instance, when the mouse pointer moves over the Inner element in this example, a `mouseover` event will be sent to that, then trickle up to Outer. This can trigger our bound `mouseover` handler at inopportune times. See the discussion for `.mouseenter()` for a useful alternative.

## Example:

Show the number of times `mouseover` and `mouseenter` events are triggered. `mouseover` fires when the pointer moves into the child element as well while `mouseenter` fires only when the pointer moves into the bound element.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5   div.out { width:40%; height:120px; margin:0 10px;
6             background-color:#D6EDFC; float:left; }
7   div.in  { width:60%; height:60%;
8             background-color:#FFCC00; margin:10px 0 0 10px; }
9   p { line-height:1em; margin:0; padding:0; }
10 </style>
11   <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
12 </head>
13 <body>
14
15 <div class="out overout">
16   <span>move your mouse</span>
17   <div class="in">
18     </div>
19 </div>
20
21 <div class="out enterleave">
22   <span>move your mouse</span>
23   <div class="in">
24     </div>
25 </div>
26
27 <script>
```

```
28 var i = 0;
29 $("div.overout").mouseover(function() {
30     i += 1;
31     $(this).find("span").text("mouse over x " + i );
32 }).mouseout(function(){
33     $(this).find("span").text("mouse out ");
34 });
35
36 var n = 0;
37 $("div.enterleave").mouseenter(function() {
38     n += 1;
39     $(this).find("span").text("mouse enter x "
40 }).mouseleave(function() {
41     $(this).find("span").text("mouse leave");
42 });
43
44 </script>
45
46 </body>
47 </html>
```

Demo

POWERED BY HERONOTE

A new version of this book is available!

---

# .mouseup()

Categories: [Events](#) > [Mouse Events](#)

`.mouseup( handler(eventObject) )`

Returns: [jQuery](#)

**Description:** *Bind an event handler to the "mouseup" JavaScript event, or trigger that event on an element.*

**.mouseup( handler(eventObject) )**

version added: [1.0](#)

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute each time the event is triggered.

**.mouseup( [eventData ],  
handler(eventObject) )**

version added: [1.4.3](#)

**eventData**

Type: [PlainObject](#)

An object containing data that will be passed to the event handler.

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute each time the event is triggered.

**.mouseup()**

version added: [1.0](#)

This method does not accept any arguments.

This method is a shortcut for `.on('mouseup', handler)` in the first variation, and `.trigger('mouseup')` in the second.

The `mouseup` event is sent to an element when the mouse pointer is over the element, and the mouse button is released. Any HTML element can receive this event.

For example, consider the HTML:

```
1 | <div id="target">
2 |   Click here
3 | </div>
```

```
4 | <div id="other">
5 |   Trigger the handler
6 | </div>
```

Click here

Trigger the handler

The event handler can be bound to any `<div>`:

```
1 | $('#target').mouseup(function() {
2 |   alert('Handler for .mouseup() called.');
```

```
3 | });
```

Now if we click on this element, the alert is displayed:

Handler for .mouseup() called.

We can also trigger the event when a different element is clicked:

```
1 | $('#other').click(function() {
2 |   $('#target').mouseup();
3 | });
```

After this code executes, clicks on Trigger the handler will also alert the message.

If the user clicks outside an element, drags onto it, and releases the button, this is still counted as a `mouseup` event. This sequence of actions is not treated as a button press in most user interfaces, so it is usually better to use the `click` event unless we know that the `mouseup` event is preferable for a particular situation.

## Example:

*Show texts when mouseup and mousedown event triggering.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-1.10.2.min.js"></script>
5 </head>
6 <body>
7   <p>Press mouse and release here.</p>
8
9   <script>
10    $("p").mouseup(function(){
11      $(this).append('<span style="color:#F00">Mouse up!</span>');
12    }).mousedown(function(){
13      $(this).append('<span style="color:#00F">Mouse down!</span>');
14    });
15
16  </script>
17
18 </body>
19 </html>
```

Demo

**A new version of this book is available!**

---

# Multiple Attribute Selector

**[name="value"]**

**[name2="value2"]**

Categories: [Selectors](#) > [Attribute](#)

## attributeMultiple selector

**Description:** *Matches elements that match all of the specified attribute filters.*

```
jQuery( "[attributeFilter1] [attributeFilter2][attributeFilterN]" )
```

version added: [1.0](#)

**attributeFilter1:** An attribute filter.

**attributeFilter2:** Another attribute filter, reducing the selection even more

**attributeFilterN:** As many more attribute filters as necessary

## Example:

*Finds all inputs that have an id attribute and whose name attribute ends with man and sets the value.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
5 </head>
6 <body>
7   <input id="man-news" name="man-news" />
8
9   <input name="milkman" />
10  <input id="letterman" name="new-letterman" />
11  <input name="newmilk" />
12
13  <script>$( 'input[id][name$="man"]' ).val( 'only' );
14
15 </body>
16 </html>
```

Demo

POWERED BY HERONOTE

A new version of this book is available!

---

# Multiple Selector (“selector1, selector2, selectorN”)

Categories: [Selectors](#) > [Basic](#)

## multiple selector

**Description:** *Selects the combined results of all the specified selectors.*

**jQuery( "selector1, selector2, selectorN" )**

version added: [1.0](#)

**selector1:** Any valid selector.

**selector2:** Another valid selector.

**selectorN:** As many more valid selectors as you like.

You can specify any number of selectors to combine into a single result. This multiple expression combinator is an efficient way to select disparate elements. The order of the DOM elements in the returned jQuery object may not be identical, as they will be in document order. An alternative to this combinator is the [.add\(\)](#) method.

## Examples:

**Example:** Finds the elements that match any of these three selectors.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5
6   div, span, p {
7     width: 126px;
8     height: 60px;
9     float:left;
10    padding: 3px;
11    margin: 2px;
12    background-color: #EEEEEE;
13    font-size:14px;
14  }
15 </style>
16 <script src="http://code.jquery.com/jquery-".
17 </head>
18 <body>
19   <div>div</div>
20
21   <p class="myClass">p class="myClass"</p>
22   <p class="notMyClass">p class="notMyClass"<
23   <span>span</span>
24   <script>$("div, span, p.myClass").css("border",
25
26 </body>
27 </html>
```

## Demo

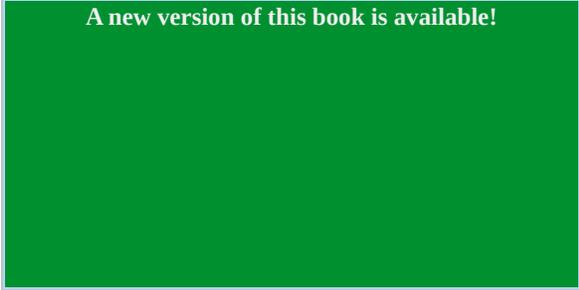
**Example:** *Show the order in the jQuery object.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5    b { color:red; font-size:16px; display:blo
6    div, span, p { width: 40px; height: 40px; flo
7                margin: 10px; background-color
8                padding:3px; color:white;
9                }
10   </style>
11   <script src="http://code.jquery.com/jquery-
12 </head>
13 <body>
14   <span>span</span>
15
16   <p>p</p>
17   <p>p</p>
18   <div>div</div>
19   <span>span</span>
20
21   <p>p</p>
22   <div>div</div>
23   <b></b>
24 <script>
25   var list = $("div,p,span").map(function (
26     return this.tagName;
27   }).get().join(", ");
28   $("b").append(document.createTextNode(lis
29 </script>
30
31 </body>
32 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

# .next()

Categories: [Traversing](#) > [Tree Traversal](#)

`.next( [selector ] )`

Returns: [jQuery](#)

**Description:** *Get the immediately following sibling of each element in the set of matched elements. If a selector is provided, it retrieves the next sibling only if it matches that selector.*

`.next( [selector ] )`

version added: [1.0](#)

**selector**

Type: [Selector](#)

A string containing a selector expression to match elements against.

Given a jQuery object that represents a set of DOM elements, the `.next()` method allows us to search through the immediately following sibling of these elements in the DOM tree and construct a new jQuery object from the matching elements.

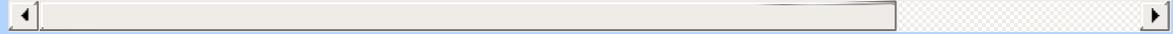
The method optionally accepts a selector expression of the same type that we can pass to the `$( )` function. If the immediately following sibling matches the selector, it remains in the newly constructed jQuery object; otherwise, it is excluded.

Consider a page with a simple list on it:

```
1 | <ul>
2 |   <li>list item 1</li>
3 |   <li>list item 2</li>
4 |   <li class="third-item">list item 3</li>
5 |   <li>list item 4</li>
6 |   <li>list item 5</li>
7 | </ul>
```

If we begin at the third item, we can find the element which comes just after it:

```
1 | $('li.third-item').next().css('background-color')
```



The result of this call is a red background behind item 4. Since we do not supply a selector expression, this following element is unequivocally included as part of the object. If we had supplied one, the element would be tested for a match before it was included.

## Examples:

**Example:** Find the very next sibling of each disabled button and change its text "this button is disabled".

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5
6   span { color:blue; font-weight:bold; }
7   button { width:100px; }
8 </style>
9   <script src="http://code.jquery.com/jquery-1.10.2.js"></script>
10 </head>
11 <body>
12   <div><button disabled="disabled">First</button>
13   <div><button>Second</button> - <span></span>
14
15   <div><button disabled="disabled">Third</button>
16   <script>$("button[disabled]").next().text("this button is disabled");</script>
17
18 </body>
19 </html>
```

Demo

**Example:** Find the very next sibling of each paragraph. Keep only the ones with a class "selected".

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-1.10.2.js"></script>
```

```
5 </head>
6 <body>
7   <p>Hello</p>
8
9   <p class="selected">Hello Again</p>
10  <div><span>And Again</span></div>
11  <script>$("p").next(".selected").css("background"
12
13 </body>
14 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# Next Adjacent Selector (“prev + next”)

Categories: [Selectors](#) > [Hierarchy](#)

## next adjacent selector

**Description:** *Selects all next elements matching "next" that are immediately preceded by a sibling "prev".*

**jQuery( "prev + next" )**

version added: [1.0](#)

**prev:** Any valid selector.

**next:** A selector to match the element that is next to the first selector.

One important point to consider with both the next adjacent sibling selector (`prev + next`) and the general sibling selector (`prev ~ siblings`) is that the elements on either side of the combinator must share the same parent.

## Example:

*Finds all inputs that are next to a label.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-
5 </head>
6 <body>
7   <form>
8
9   <label>Name:</label>
10  <input name="name" />
11  <fieldset>
12    <label>Newsletter:</label>
13
14    <input name="newsletter" />
15  </fieldset>
16 </form>
17 <input name="none" />
18 <script>$("label + input").css("color", "blue
19
20 </body>
21 </html>
```

Demo

**A new version of this book is available!**

---

# Next Siblings Selector (“prev ~ siblings”)

Categories: [Selectors](#) > [Hierarchy](#)

## next siblings selector

**Description:** *Selects all sibling elements that follow after the "prev" element, have the same parent, and match the filtering "siblings" selector.*

**jQuery( "prev ~ siblings" )**

version added: [1.0](#)

**prev:** Any valid selector.

**siblings:** A selector to filter elements that are the following siblings of the first selector.

The notable difference between ( `prev + next` ) and ( `prev ~ siblings` ) is their respective reach. While the former reaches only to the immediately following sibling element, the latter extends that reach to all following sibling elements.

## Example:

*Finds all divs that are siblings after the element with #prev as its id. Notice the span isn't selected since it is not a div and the "niece" isn't selected since it is a child of a sibling, not an actual sibling.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5
6   div, span {
7     display: block;
8     width: 80px;
9     height: 80px;
10    margin: 5px;
11    background: #bbffaa;
12    float: left;
13    font-size: 14px;
14  }
15  div#small {
16    width: 60px;
17    height: 25px;
18    font-size: 12px;
19    background: #fab;
20  }
21 </style>
22 <script src="http://code.jquery.com/jquery-
23 </head>
24 <body>
25   <div>div (doesn't match since before #prev)
26   <span id="prev">span#prev</span>
27   <div>div sibling</div>
28
```

```
29 <div>div sibling <div id="small">div niece</div>
30 <span>span sibling (not div)</span>
31 <div>div sibling</div>
32 <script>$("#prev ~ div").css("border", "3px groove blue");
33
34 </body>
35 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# .nextAll()

Categories: [Traversing](#) > [Tree Traversal](#)

`.nextAll( [selector ] )`

Returns: [jQuery](#)

**Description:** *Get all following siblings of each element in the set of matched elements, optionally filtered by a selector.*

`.nextAll( [selector ] )`

version added: [1.2](#)

**selector**

Type: [String](#)

A string containing a selector expression to match elements against.

Given a jQuery object that represents a set of DOM elements, the `.nextAll()` method allows us to search through the successors of these elements in the DOM tree and construct a new jQuery object from the matching elements.

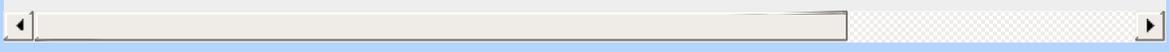
The method optionally accepts a selector expression of the same type that we can pass to the `$( )` function. If the selector is supplied, the elements will be filtered by testing whether they match it.

Consider a page with a simple list on it:

```
1 | <ul>
2 |     <li>list item 1</li>
3 |     <li>list item 2</li>
4 |     <li class="third-item">list item 3</li>
5 |     <li>list item 4</li>
6 |     <li>list item 5</li>
7 | </ul>
```

If we begin at the third item, we can find the elements which come after it:

```
1 | $('li.third-item').nextAll().css('background-c
```



The result of this call is a red background behind items 4 and 5. Since we do not supply a selector expression, these following elements are unequivocally included as part of the object. If we had supplied one, the elements would be tested for a match before they were included.

## Examples:

**Example:** *Locate all the divs after the first and give them a class.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     div { width: 80px; height: 80px; background
6           border: 2px solid black; margin: 10px
7     div.after { border-color: red; }
8   </style>
9   <script src="http://code.jquery.com/jquery-
10 </head>
11 <body>
12   <div>first</div>
13   <div>sibling<div>child</div></div>
14   <div>sibling</div>
15
16   <div>sibling</div>
17 <script>$("div:first").nextAll().addClass("af
18
19 </body>
20 </html>
```

Demo

**Example:** *Locate all the paragraphs after the second child in the body and give them a class.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
```

```
4 <style>
5 div, p { width: 60px; height: 60px; background:
6         border: 2px solid black; margin: 10px;
7 .after { border-color: red; }
8 </style>
9 <script src="http://code.jquery.com/jquery-latest.js"
10 </head>
11 <body>
12 <p>p</p>
13 <div>div</div>
14 <p>p</p>
15 <p>p</p>
16 <div>div</div>
17
18 <p>p</p>
19 <div>div</div>
20 <script>
21     $(":nth-child(1)").nextAll("p").addClass("after"
22 </script>
23
24 </body>
25 </html>
```

Demo

POWERED BY HERONOTE

A new version of this book is available!

---

# .nextUntil()

Categories: [Traversing](#) > [Tree Traversal](#)

`.nextUntil( [selector ] [, filter ] )`

Returns: [jQuery](#)

**Description:** *Get all following siblings of each element up to but not including the element matched by the selector, DOM node, or jQuery object passed.*

`.nextUntil( [selector ] [, filter ] )`

version added: [1.4](#)

**selector**

Type: [Selector](#)

A string containing a selector expression to indicate where to stop matching following sibling elements.

**filter**

Type: [Selector](#)

A string containing a selector expression to match elements against.

`.nextUntil( [element ] [, filter ] )`

version added: [1.6](#)

**element**

Type: [Element](#)

A DOM node or jQuery object indicating where to stop matching following sibling elements.

**filter**

Type: [Selector](#)

A string containing a selector expression to match elements against.

Given a selector expression that represents a set of DOM elements, the `.nextUntil()` method searches through the successors of these elements in the DOM tree, stopping when it reaches an element matched by the method's argument. The new jQuery object that is returned contains all following siblings up to but not including the one matched by the `.nextUntil()` argument.

If the selector is not matched or is not supplied, all following siblings will be selected; in these cases it selects the same elements as the `.nextAll()` method does when no filter selector is provided.

**As of jQuery 1.6**, A DOM node or jQuery object, instead of a selector, may be passed to the `.nextUntil()` method.

The method optionally accepts a selector expression for its second argument. If this argument is supplied, the elements will be filtered by testing whether they match it.

## Example:

*Find the siblings that follow `<dt id="term-2">` up to the next `<dt>` and give them a red background color. Also find `<dd>` siblings that follow `<dt id="term-1">` up to `<dt id="term-3">` and give them a green text color.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-1.10.2.min.js"></script>
5 </head>
6 <body>
7   <dl>
8     <dt id="term-1">term 1</dt>
9     <dd>definition 1-a</dd>
10    <dd>definition 1-b</dd>
11    <dd>definition 1-c</dd>
12    <dd>definition 1-d</dd>
13
14    <dt id="term-2">term 2</dt>
15    <dd>definition 2-a</dd>
16    <dd>definition 2-b</dd>
17    <dd>definition 2-c</dd>
18
19    <dt id="term-3">term 3</dt>
20    <dd>definition 3-a</dd>
21    <dd>definition 3-b</dd>
22  </dl>
23  <script>
24    $("#term-2").nextUntil("dt")
25      .css("background-color", "red");
26
27    var term3 = document.getElementById("term-3")
28    $("#term-1").nextUntil(term3, "dd")
```

```
29     .css("color", "green");
30
31 </script>
32
33 </body>
34 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# .not()

Categories: [Traversing](#) > [Filtering](#) | [Traversing](#) > [Miscellaneous Traversing](#)

`.not( selector )`

Returns: [jQuery](#)

**Description:** *Remove elements from the set of matched elements.*

**.not( selector )**

version added: [1.0](#)

**selector**

Type: [Selector](#)

A string containing a selector expression to match elements against.

**.not( elements )**

version added: [1.0](#)

**elements**

Type: [Elements](#)

One or more DOM elements to remove from the matched set.

**.not( function(index) )**

version added: [1.4](#)

**function(index)**

Type: [Function\(\)](#)

A function used as a test for each element in the set. `this` is the current DOM element.

**.not( jQuery object )**

version added: [1.4](#)

**jQuery object**

Type: [PlainObject](#)

An existing jQuery object to match the current set of elements against.

Given a jQuery object that represents a set of DOM elements, the `.not()` method constructs a new jQuery object from a subset of the matching elements. The supplied selector is tested against each element; the elements that don't match the selector will be included in the result.

Consider a page with a simple list on it:

```
1 | <ul>
2 |   <li>list item 1</li>
3 |   <li>list item 2</li>
4 |   <li>list item 3</li>
5 |   <li>list item 4</li>
6 |   <li>list item 5</li>
7 | </ul>
```

We can apply this method to the set of list items:

```
1 | $('li').not(':even').css('background-color',
```

The result of this call is a red background for items 2 and 4, as they do not match the selector (recall that `:even` and `:odd` use 0-based indexing).

## Removing Specific Elements

The second version of the `.not()` method allows us to remove elements from the matched set, assuming we have found those elements previously by some other means. For example, suppose our list had an id applied to one of its items:

```
1 | <ul>
2 |   <li>list item 1</li>
3 |   <li>list item 2</li>
4 |   <li id="notli">list item 3</li>
5 |   <li>list item 4</li>
6 |   <li>list item 5</li>
7 | </ul>
```

We can fetch the third list item using the native JavaScript `getElementById()` function, then remove it from a jQuery object:

```
1 | $('li').not(document.getElementById('notli'))  
2 |   .css('background-color', 'red');
```

This statement changes the color of items 1, 2, 4, and 5. We could have accomplished the same thing with a simpler jQuery expression, but this technique can be useful when, for example, other libraries provide references to plain DOM nodes.

As of jQuery 1.4, the `.not()` method can take a function as its argument in the same way that `.filter()` does. Elements for which the function returns `true` are excluded from the filtered set; all other elements are included.

## Examples:

**Example:** *Adds a border to divs that are not green or blue.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5   div { width:50px; height:50px; margin:10px; f
6         background:yellow; border:2px solid whi
7   .green { background:#8f8; }
8   .gray { background:#ccc; }
9   #blueone { background:#99f; }
10  </style>
11    <script src="http://code.jquery.com/jquery-
12  </head>
13  <body>
14    <div></div>
15    <div id="blueone"></div>
16    <div></div>
17    <div class="green"></div>
18
19    <div class="green"></div>
20    <div class="gray"></div>
21    <div></div>
22
23    <script>
24    $("div").not(".green, #blueone")
25              .css("border-color", "red");
26
27    </script>
28
29  </body>
30 </html>
```

## Demo

**Example** • Removes the element with the ID "selected" from the set of all paragraphs.

```
1 | $("p").not( $("#selected")[0] )
```

**Example** • Removes the element with the ID "selected" from the set of all paragraphs.

```
1 | $("p").not( "#selected" )
```

**Example** • Removes all elements that match "div p.selected" from the total set of all paragraphs.

```
1 | $("p").not( $("div p.selected" ) )
```

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# :not() Selector

Categories: [Selectors](#) > [Basic Filter](#)

## not selector

**Description:** *Selects all elements that do not match the given selector.*

**jQuery( ":not(selector)" )**

version added: [1.0](#)

**selector:** A selector with which to filter by.

All selectors are accepted inside `:not()`, for example: `:not(div a)` and `:not(div, a)`.

## Additional Notes

The `.not()` method will end up providing you with more readable selections than pushing complex selectors or variables into a `:not()` selector filter. In most cases, it is a better choice.

## Example:

*Finds all inputs that are not checked and highlights the next sibling span. Notice there is no change when clicking the checkboxes since no click events have been linked.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <script src="http://code.jquery.com/jquery-1.10.2.min.js"></script>
5  </head>
6  <body>
7    <div>
8      <input type="checkbox" name="a" />
9      <span>Mary</span>
10   </div>
11
12   <div>
13     <input type="checkbox" name="b" />
14     <span>lcm</span>
15
16   </div>
17   <div>
18     <input type="checkbox" name="c" checked="" />
19     <span>Peter</span>
20   </div>
21
22   <script>
23     $("input:not(:checked) + span").css("background-color", "#f0f0f0");
24     $("input").attr("disabled", "disabled");
25
26   </script>
27
28 </body>
```

29 | `</html>`

Demo

POWERED BY **HERONOTE**

A new version of this book is available!

---

# :nth-child() Selector

Categories: [Selectors](#) > [Child Filter](#)

## nth-child selector

**Description:** *Selects all elements that are the nth-child of their parent.*

**jQuery( ":nth-child(index/even/odd/equation)" )**

version added: [1.1.4](#)

**index:** The index of each child to match, starting with **1**, the string **even** or **odd**, or an equation ( eg. `:nth-child(even)`, `:nth-child(4n)` )

Because jQuery's implementation of `:nth-` selectors is strictly derived from the CSS specification, the value of `n` is "1-indexed", meaning that the counting starts at 1. For other selector expressions such as `:eq()` or `:even` jQuery follows JavaScript's "0-indexed" counting. Given a single `<ul>` containing two `<li>`s, `$('li:nth-child(1)')` selects the first `<li>` while `$('li:eq(1)')` selects the second.

The `:nth-child(n)` pseudo-class is easily confused with `:eq(n)`, even though the two can result in dramatically different matched elements. With `:nth-child(n)`, all children are counted, regardless of what they are, and the specified element is selected only if it matches the selector attached to the pseudo-class. With `:eq(n)` only the selector attached to the pseudo-class is counted, not limited to children of any other element, and the (n+1)th one (n is 0-based) is selected.

Further discussion of this unusual usage can be found in the [W3C CSS specification](#).

## Examples:

**Example:** Find the second li in each matched ul and note it.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     div { float:left; }
6     span { color:blue; }
7   </style>
8   <script src="http://code.jquery.com/jquery-
9 </head>
10 <body>
11   <div>
12     <ul>
13       <li>John</li>
14       <li>Karl</li>
15       <li>Brandon</li>
16
17     </ul>
18   </div>
19   <div>
20     <ul>
21       <li>Sam</li>
22     </ul>
23   </div>
24   <div>
25     <ul>
26       <li>Glen</li>
27       <li>Tane</li>
28       <li>Ralph</li>
29       <li>David</li>
30     </ul>
```

```
31 </div>
32 <script>$("ul li:nth-child(2)").append("<span> - 2nd!</>");
33
34 </body>
35 </html>
```

## Demo

**Example:** This is a playground to see how the selector works with different strings. Notice that this is different from the `even` and `odd` which have no regard for parent and just filter the list of elements to every other one. The `nth-child` however counts the index of the child to its particular parent. In any case, it's easier to see than explain so...

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     button { display:block; font-size:12px; width
6     div { float:left; margin:10px; font-size:10px
7         border:1px solid black; }
8     span { color:blue; font-size:18px; }
9     #inner { color:red; }
10    td { width:50px; text-align:center; }
11  </style>
12    <script src="http://code.jquery.com/jquery-
13  </head>
14  <body>
15    <div>
16      <button>:nth-child(even)</button>
17      <button>:nth-child(odd)</button>
18      <button>:nth-child(3n)</button>
19      <button>:nth-child(2)</button>
20    </div>
21    <div>
```

```
22     <button>:nth-child(3n+1)</button>
23     <button>:nth-child(3n+2)</button>
24     <button>:even</button>
25     <button>:odd</button>
26 </div>
27
28 <div>
29     <table>
30         <tr><td>John</td></tr>
31         <tr><td>Karl</td></tr>
32         <tr><td>Brandon</td></tr>
33         <tr><td>Benjamin</td></tr>
34     </table>
35 </div>
36 <div>
37     <table>
38         <tr><td>Sam</td></tr>
39     </table>
40 </div>
41 <div>
42     <table>
43         <tr><td>Glen</td></tr>
44         <tr><td>Tane</td></tr>
45         <tr><td>Ralph</td></tr>
46         <tr><td>David</td></tr>
47         <tr><td>Mike</td></tr>
48         <tr><td>Dan</td></tr>
49     </table>
50 </div>
51
52 <span>tr<span id="inner"></span></span>
53
54 <script>
55 $( "button" ).click( function () {
56     var str = $( this ).text();
57     $( "tr" ).css( "background", "white" );
58     $( "tr" + str ).css( "background", "#ff0000" );
```

```
59     $("#inner").text(str);
60   });
61 </script>
62
63 </body>
64 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# **:nth-last-child() Selector**

Categories: [Selectors](#) > [Child Filter](#)

## nth-last-child selector

**Description:** *Selects all elements that are the nth-child of their parent, counting from the last element to the first.*

**jQuery( ":nth-last-child(index/even/odd/equation)" )**

version added: [1.9](#)

**index:** The index of each child to match, starting with the last one (1), the string `even` or `odd`, or an equation ( eg. `:nth-last-child(even)`, `:nth-last-child(4n)` )

Because jQuery's implementation of `:nth-` selectors is strictly derived from the CSS specification, the value of `n` is "1-indexed", meaning that the counting starts at 1. For other selector expressions such as `:eq()` or `:even` jQuery follows JavaScript's "0-indexed" counting. Given a single `<ul>` containing three `<li>S`, `$('.li:nth-last-child(1)')` selects the third, last, `<li>`.

Further discussion of this usage can be found in the [W3C CSS specification](#).

## Examples:

**Example:** Find the second to last li in each matched ul and note it.

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5    div { float:left; }
6    span { color:blue; }
7  </style>
8  <script src="http://code.jquery.com/jquery-
9  </head>
10 <body>
11   <div>
12     <ul>
13       <li>John</li>
14       <li>Karl</li>
15       <li>Adam</li>
16     </ul>
17   </div>
18   <div>
19     <ul>
20       <li>Dan</li>
21     </ul>
22   </div>
23   <div>
24     <ul>
25       <li>Dave</li>
26       <li>Rick</li>
27       <li>Timmy</li>
28       <li>Gibson</li>
29     </ul>
30   </div>
```

```
31 <script>$("ul li:nth-last-child(2)").append("<span> - 2")
32
33 </body>
34 </html>
```

## Demo

**Example:** *This is a nlayout to see how the selector works with different strings.*

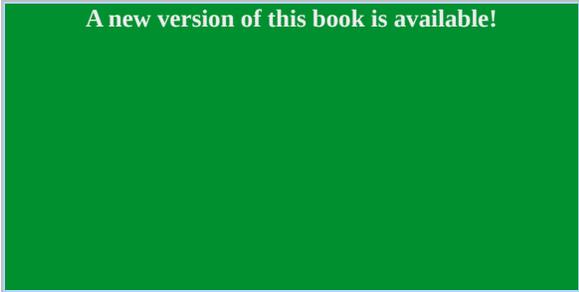
```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <style>
5 button { display:block; font-size:12px; width
6 div { float:left; margin:10px; font-size:10px
7 border:1px solid black; }
8 span { color:blue; font-size:18px; }
9 #inner { color:red; }
10 td { width:50px; text-align:center; }
11 </style>
12 <script src="http://code.jquery.com/jquery-
13 </head>
14 <body>
15 <div>
16 <button>:nth-last-child(even)</button>
17 <button>:nth-last-child(odd)</button>
18 <button>:nth-last-child(3n)</button>
19 <button>:nth-last-child(2)</button>
20 </div>
21 <div>
22 <button>:nth-last-child(3n+1)</button>
23 <button>:nth-last-child(3n+2)</button>
24 </div>
25
26 <div>
```

```
27     <table>
28         <tr><td>John</td></tr>
29         <tr><td>Karl</td></tr>
30         <tr><td>Brandon</td></tr>
31         <tr><td>Benjamin</td></tr>
32     </table>
33 </div>
34 <div>
35     <table>
36         <tr><td>Sam</td></tr>
37     </table>
38 </div>
39 <div>
40     <table>
41         <tr><td>Glen</td></tr>
42         <tr><td>Tane</td></tr>
43         <tr><td>Ralph</td></tr>
44         <tr><td>David</td></tr>
45         <tr><td>Mike</td></tr>
46         <tr><td>Dan</td></tr>
47     </table>
48 </div>
49
50 <span>tr<span id="inner"></span></span>
51
52 <script>
53 $( "button" ).click( function ( ) {
54     var str = $( this ).text();
55     $( "tr" ).css( "background", "white" );
56     $( "tr" + str ).css( "background", "#ff0000" );
57     $( "#inner" ).text( str );
58 });
59 </script>
60
61 </body>
62 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

# :nth-last-of-type() Selector

Categories: [Selectors](#) > [Child Filter](#)

## nth-last-of-type selector

**Description:** *Selects all elements that are the nth-child of their parent, counting from the last element to the first.*

**jQuery( ":nth-last-of-type(index/even/odd/equation)" )**

version added: [1.9](#)

**index:** The index of each child to match, starting with the last one (1), the string `even` or `odd`, or an equation ( eg. `:nth-last-of-type(even)`, `:nth-last-of-type(4n)` )

Because jQuery's implementation of `:nth-` selectors is strictly derived from the CSS specification, the value of `n` is "1-indexed", meaning that the counting starts at 1. For other selector expressions such as `:eq()` or `:even` jQuery follows JavaScript's "0-indexed" counting. Given a single `<ul>` containing three `<li>S`, `$( 'li:nth-last-of-type(1)' )` selects the third, last, `<li>`.

Further discussion of this usage can be found in the [W3C CSS specification](#).

## Examples:

**Example:** Find the second to last li in each matched ul and note it.

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5    div { float:left; }
6    span { color:blue; }
7  </style>
8  <script src="http://code.jquery.com/jquery-
9  </head>
10 <body>
11   <div>
12     <ul>
13       <li>John</li>
14       <li>Karl</li>
15       <li>Adam</li>
16     </ul>
17   </div>
18   <div>
19     <ul>
20       <li>Dan</li>
21     </ul>
22   </div>
23   <div>
24     <ul>
25       <li>Dave</li>
26       <li>Rick</li>
27       <li>Timmy</li>
28       <li>Gibson</li>
29     </ul>
30   </div>
```

```
31 <script>$("ul li:nth-last-of-type(2)").append("<span> -  
32  
33 </body>  
34 </html>
```

## Demo

**Example:** *This is a nlayout to see how the selector works with different strings.*

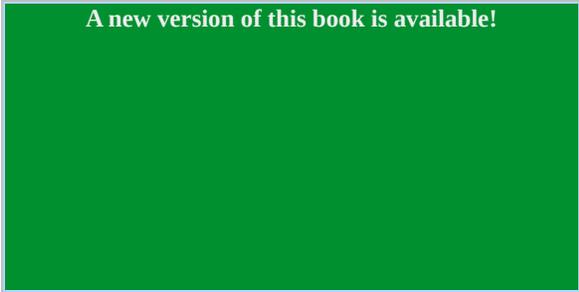
```
1 <!DOCTYPE html>  
2 <html>  
3 <head>  
4 <style>  
5 button { display:block; font-size:12px; width  
6 div { float:left; margin:10px; font-size:10px  
7 border:1px solid black; }  
8 span { color:blue; font-size:18px; }  
9 #inner { color:red; }  
10 td { width:50px; text-align:center; }  
11 </style>  
12 <script src="http://code.jquery.com/jquery-  
13 </head>  
14 <body>  
15 <div>  
16 <button>nth-last-of-type(even)</button>  
17 <button>nth-last-of-type(odd)</button>  
18 <button>nth-last-of-type(3n)</button>  
19 <button>nth-last-of-type(2)</button>  
20 </div>  
21 <div>  
22 <button>nth-last-of-type(3n+1)</button>  
23 <button>nth-last-of-type(3n+2)</button>  
24 </div>  
25  
26 <div>
```

```
27     <table>
28         <tr><td>John</td></tr>
29         <tr><td>Karl</td></tr>
30         <tr><td>Brandon</td></tr>
31         <tr><td>Benjamin</td></tr>
32     </table>
33 </div>
34 <div>
35     <table>
36         <tr><td>Sam</td></tr>
37     </table>
38 </div>
39 <div>
40     <table>
41         <tr><td>Glen</td></tr>
42         <tr><td>Tane</td></tr>
43         <tr><td>Ralph</td></tr>
44         <tr><td>David</td></tr>
45         <tr><td>Mike</td></tr>
46         <tr><td>Dan</td></tr>
47     </table>
48 </div>
49
50 <span>tr<span id="inner"></span></span>
51
52 <script>
53 $( "button" ).click( function ( ) {
54     var str = $( this ).text();
55     $( "tr" ).css( "background", "white" );
56     $( "tr" + str ).css( "background", "#ff0000" );
57     $( "#inner" ).text( str );
58 });
59 </script>
60
61 </body>
62 </html>
```

Demo

POWERED BY HERONOTE

A new version of this book is available!



---

# :nth-of-type() Selector

Categories: [Selectors](#) > [Child Filter](#)

## nth-of-type selector

**Description:** *Selects all elements that are the nth child of their parent in relation to siblings with the same element name.*

jQuery( ":nth-of-type(index/even/odd/equation)" )

version added: [1.9](#)

**index:** The index of each child to match, starting with [1](#), the string [even](#) or [odd](#), or an equation ( eg. [:nth-of-type\(even\)](#), [:nth-of-type\(4n\)](#) )

Because jQuery's implementation of [:nth-](#) selectors is strictly derived from the CSS specification, the value of [n](#) is "1-indexed", meaning that the counting starts at 1. For other selector expressions such as [:eq\(\)](#) or [:even](#) jQuery follows JavaScript's "0-indexed" counting.

Further discussion of this usage can be found in the [W3C CSS specification](#).

## Example:

*Find each span that is second in relation to its sibling spans.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     .nth { color: red; }
6   </style>
7   <script src="http://code.jquery.com/jquery-
8 </head>
9 <body>
10  <div>
11    <span>John</span>,
12    <b>Kim</b>,
13    <span>Adam</span>,
14    <b>Rafael</b>,
15    <span>Oleg</span>
16  </div>
17  <div>
18    <b>Dave</b>,
19    <span>Ann</span>
20  </div>
21  <div>
22    <i><span>Maurice</span></i>,
23    <span>Richard</span>,
24    <span>Ralph</span>,
25    <span>Jason</span>
26  </div>
27  <script>
28    $( "span:nth-of-type(2)" )
29      .append( "<span> is 2nd sibling span</span>" )
30      .addClass( "nth" );
```

```
31 </script>
32
33 </body>
34 </html>
```

Demo

POWERED BY **HERONOTE**

A new version of this book is available!

---

# :odd Selector

Categories: [Selectors](#) > [Basic Filter](#) | [Selectors](#) > [jQuery Extensions](#)

## odd selector

**Description:** *Selects odd elements, zero-indexed. See also [even](#).*

**jQuery( ":odd" )**

version added: [1.0](#)

In particular, note that the *0-based indexing* means that, counter-intuitively, `:odd` selects the second element, fourth element, and so on within the matched set.

### Additional Notes:

Because `:odd` is a jQuery extension and not part of the CSS specification, queries using `:odd` cannot take advantage of the performance boost provided by the native DOM `querySelectorAll()` method. To achieve the best performance when using `:odd` to select elements, first select the elements using a pure CSS selector, then use `.filter(":odd")`.

Selected elements are in the order of their appearance in the document.

## Example:

*Finds odd table rows matching the second, fourth and so on (index 1, 3, 5 etc.).*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5  table {
6    background:#f3f7f5;
7  }
8  </style>
9    <script src="http://code.jquery.com/jquery-
10 </head>
11 <body>
12   <table border="1">
13     <tr><td>Row with Index #0</td></tr>
14     <tr><td>Row with Index #1</td></tr>
15
16     <tr><td>Row with Index #2</td></tr>
17     <tr><td>Row with Index #3</td></tr>
18   </table>
19   <script>$("tr:odd").css("background-color", "
20
21 </body>
22 </html>
```

Demo

**A new version of this book is available!**

---

# .off()

Categories: [Events](#) > [Event Handler Attachment](#)

`.off( events [, selector ] [, handler(eventObject) ] )`

Returns: [jQuery](#)

**Description:** *Remove an event handler.*

**`.off( events [, selector ] [, handler(eventObject) ] )`**

version added: [1.7](#)

**events**

Type: [String](#)

One or more space-separated event types and optional namespaces, or just namespaces, such as "click", "keydown.myPlugin", or ".myPlugin".

**selector**

Type: [String](#)

A selector which should match the one originally passed to `.on()` when attaching event handlers.

**handler(eventObject)**

Type: [Function\(\)](#)

A handler function previously attached for the event(s), or the special value `false`.

**`.off( events [, selector ] )`**

version added: [1.7](#)

**events**

Type: [PlainObject](#)

An object where the string keys represent one or more space-separated event types and optional namespaces, and the values represent handler functions previously attached for the event(s).

**selector**

Type: [String](#)

A selector which should match the one originally passed to `.on()` when attaching event handlers.

The `off()` method removes event handlers that were attached with `.on()`. See the discussion of delegated and directly bound events on

that page for more information. Specific event handlers can be removed on elements by providing combinations of event names, namespaces, selectors, or handler function names. **When multiple filtering arguments are given, all of the arguments provided must match for the event handler to be removed.**

If a simple event name such as `"click"` is provided, *all* events of that type (both direct and delegated) are removed from the elements in the jQuery set. When writing code that will be used as a plugin, or simply when working with a large code base, best practice is to attach and remove events using namespaces so that the code will not inadvertently remove event handlers attached by other code. All events of all types in a specific namespace can be removed from an element by providing just a namespace, such as `".myPlugin"`. At minimum, either a namespace or event name must be provided.

To remove specific delegated event handlers, provide a `selector` argument. The selector string must exactly match the one passed to `.on()` when the event handler was attached. To remove all delegated events from an element without removing non-delegated events, use the special value `"**"`.

A handler can also be removed by specifying the function name in the `handler` argument. When jQuery attaches an event handler, it assigns a unique id to the handler function. Handlers proxied by `jQuery.proxy()` or a similar mechanism will all have the same unique id (the proxy function), so passing proxied handlers to `.off` may remove more handlers than intended. In those situations it is better to attach and remove event handlers using namespaces.

As with `.on()`, you can pass `events` as an object instead of specifying an `events` string and `handler` function as separate arguments. The keys for the `events` object are events and/or namespaces; the values are handler functions or the special value `false`.

## Examples:

**Example:** Add and remove event handlers on the colored button.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5   button { margin:5px; }
6   button#theone { color:red; background:yellow;
7 </style>
8   <script src="http://code.jquery.com/jquery-
9 </head>
10 <body>
11   <button id="theone">Does nothing...</button>
12 <button id="bind">Add Click</button>
13 <button id="unbind">Remove Click</button>
14 <div style="display:none;">Click!</div>
15 <script>
16 function aClick() {
17   $("div").show().fadeOut("slow");
18 }
19 $("#bind").click(function () {
20   $("body").on("click", "#theone", aClick)
21     .find("#theone").text("Can Click!");
22 });
23 $("#unbind").click(function () {
24   $("body").off("click", "#theone", aClick)
25     .find("#theone").text("Does nothing...");
26 });
27 </script>
28
29 </body>
30 </html>
```

## Demo

**Example** Remove all event handlers from all paragraphs:

```
1 | $("p").off()
```

**Example** Remove all delegated click handlers from all paragraphs:

```
1 | $("p").off("click", "**")
```

**Example** Remove just one previously bound handler by passing it as the third argument:

```
1 | var foo = function () {  
2 |     // code to handle some kind of event  
3 | };  
4 |  
5 | // ... now foo will be called when paragraphs  
6 | $("body").on("click", "p", foo);  
7 |  
8 | // ... foo will no longer be called.  
9 | $("body").off("click", "p", foo);
```

**Example** Unbind all delegated event handlers by their namespace:

```
1 | var validate = function () {  
2 |     // code to validate form entries  
3 | };
```

```
4 // delegate events under the ".validator" namespace
5 $("form").on("click.validator", "button", validate);
6
7
8 $("form").on("keypress.validator", "input[type='text']"
9
10 // remove event handlers in the ".validator" namespace
11
12 $("form").off(".validator");
```

POWERED BYHERONOTE

A new version of this book is available!

---

# .offset()

Categories: [CSS](#) | [Offset](#) | [Manipulation](#) > [Style Properties](#)

Get the current coordinates of the first element, or set the coordinates of every element, in the set of matched elements, relative to the document.

## Contents:

[.offset\(\)](#)

`.offset()`

[.offset\( coordinates \)](#)

`.offset( coordinates )`

`.offset( function(index, coords) )`

`.offset()`

Returns: `Object`

**Description:** *Get the current coordinates of the first element in the set of matched elements, relative to the document.*

`.offset()`

version added: `1.2`

This method does not accept any arguments.

The `.offset()` method allows us to retrieve the current position of an element *relative to the document*. Contrast this with `.position()`, which retrieves the current position *relative to the offset parent*. When positioning a new element on top of an existing one for global manipulation (in particular, for implementing drag-and-drop), `.offset()` is the more useful.

`.offset()` returns an object containing the properties `top` and `left`.

**Note:** *jQuery does not support getting the offset coordinates of hidden elements or accounting for borders, margins, or padding set on the body element.*

*While it is possible to get the coordinates of elements with `visibility:hidden` set, `display:none` is excluded from the rendering tree and thus has a position that is undefined.*

## Examples:

**Example:** *Access the offset of the second paragraph:*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5   p { margin-left:10px; }
6   </style>
7   <script src="http://code.jquery.com/jquery-...
8 </head>
9 <body>
10  <p>Hello</p><p>2nd Paragraph</p>
11  <script>var p = $("p:last");
12  var offset = p.offset();
13  p.html( "left: " + offset.left + ", top: " +
14
15 </body>
16 </html>
```

Demo

**Example:** *Click to see the offset.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5   p { margin-left:10px; color:blue; width:200px;
6     cursor:pointer; }
7   span { color:red; cursor:pointer; }
8   div.abs { width:50px; height:50px; position:abs
9     left:220px; top:35px; background-co.
```

```
10     cursor:pointer; }
11 </style>
12 <script src="http://code.jquery.com/jquery-latest.js"
13 </head>
14 <body>
15     <div id="result">Click an element.</div>
16 <p>
17     This is the best way to <span>find</span> an offset.
18 </p>
19
20 <div class="abs">
21 </div>
22
23 <script>
24 $( "*", document.body).click(function (e) {
25     var offset = $(this).offset();
26     e.stopPropagation();
27     $("#result").text(this.tagName + " coords ( "
28                       offset.top +
29 });
30
31 </script>
32
33 </body>
34 </html>
```

Demo

`.offset( coordinates )`

Returns: [jQuery](#)

**Description:** *Set the current coordinates of every element in the set of matched elements, relative to the document.*

`.offset( coordinates )`

version added: [1.4](#)

**coordinates**

Type: [PlainObject](#)

An object containing the properties `top` and `left`, which are integers indicating the new top and left coordinates for the elements.

`.offset( function(index, coords) )`

version added: [1.4](#)

**function(index, coords)**

Type: [Function\(\)](#)

A function to return the coordinates to set. Receives the index of the element in the collection as the first argument and the current coordinates as the second argument. The function should return an object with the new `top` and `left` properties.

The `.offset()` setter method allows us to reposition an element. The element's position is specified *relative to the document*. If the element's `position` style property is currently `static`, it will be set to `relative` to allow for this repositioning.

## Example:

*Set the offset of the second paragraph:*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>p { margin-left:10px; } </style>
5    <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
6  </head>
7  <body>
8    <p>Hello</p><p>2nd Paragraph</p>
9    <script>$("p:last").offset({ top: 10, left: 30});</script>
10
11 </body>
12 </html>
```

Demo

POWERED BY **HERONOTE**

A new version of this book is available!

---

# .offsetParent()

Categories: [Offset](#) | [Traversing](#) > [Tree Traversal](#)

`.offsetParent()`

Returns: [jQuery](#)

**Description:** *Get the closest ancestor element that is positioned.*

`.offsetParent()`

version added: [1.2.6](#)

This method does not accept any arguments.

Given a jQuery object that represents a set of DOM elements, the `.offsetParent()` method allows us to search through the ancestors of these elements in the DOM tree and construct a new jQuery object wrapped around the closest positioned ancestor. An element is said to be positioned if it has a CSS position attribute of `relative`, `absolute`, or `fixed`. This information is useful for calculating offsets for performing animations and placing objects on the page.

Consider a page with a basic nested list on it, with a positioned element:

```
1 <ul class="level-1">
2   <li class="item-i">I</li>
3   <li class="item-ii" style="position: relat:
4     <ul class="level-2">
5       <li class="item-a">A</li>
6       <li class="item-b">B
7         <ul class="level-3">
8           <li class="item-1">1</li>
9           <li class="item-2">2</li>
10          <li class="item-3">3</li>
11        </ul>
12      </li>
13      <li class="item-c">C</li>
14    </ul>
15  </li>
16  <li class="item-iii">III</li>
```

```
17 | </ul>
```

If we begin at item A, we can find its positioned ancestor:

```
1 | $('li.item-a').offsetParent().css('background-
```

This will change the color of list item II, which is positioned.

## Example:

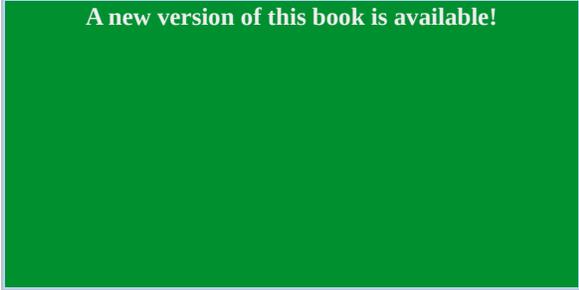
*Find the offsetParent of item "A."*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-1.10.2.min.js"></script>
5 </head>
6 <body>
7
8 <ul class="level-1">
9   <li class="item-i">I</li>
10  <li class="item-ii" style="position: relative;">
11    <ul class="level-2">
12      <li class="item-a">A</li>
13      <li class="item-b">B
14        <ul class="level-3">
15          <li class="item-1">1</li>
16          <li class="item-2">2</li>
17          <li class="item-3">3</li>
18        </ul>
19      </li>
20      <li class="item-c">C</li>
21    </ul>
22  </li>
23  <li class="item-iii">III</li>
24 </ul>
25 <script>$('li.item-a').offsetParent().css('background-color','red');</script>
26
27 </body>
28 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

# .on()

Categories: [Events](#) > [Event Handler Attachment](#)

`.on( events [, selector ] [, data ], handler(eventObject) )`

Returns: [jQuery](#)

**Description:** *Attach an event handler function for one or more events to the selected elements.*

`.on( events [, selector ] [, data ], handler(eventObject) )`

version added: [1.7](#)

#### events

Type: [String](#)

One or more space-separated event types and optional namespaces, such as "click" or "keydown.myPlugin".

#### selector

Type: [String](#)

A selector string to filter the descendants of the selected elements that trigger the event. If the selector is `null` or omitted, the event is always triggered when it reaches the selected element.

#### data

Type: [Anything](#)

Data to be passed to the handler in `event.data` when an event is triggered.

#### handler(eventObject)

Type: [Function\(\)](#)

A function to execute when the event is triggered. The value `false` is also allowed as a shorthand for a function that simply does `return false`.

`.on( events [, selector ] [, data ] )`

version added: [1.7](#)

#### events

Type: [PlainObject](#)

An object in which the string keys represent one or more space-separated event types and optional namespaces, and the values represent a handler function to be called for the event(s).

**selector**Type: [String](#)

A selector string to filter the descendants of the selected elements that will call the handler. If the selector is null or omitted, the handler is always called when it reaches the selected element.

**data**Type: [Anything](#)

Data to be passed to the handler in `event.data` when an event occurs.

The `.on()` method attaches event handlers to the currently selected set of elements in the jQuery object. As of jQuery 1.7, the `.on()` method provides all functionality required for attaching event handlers. For help in converting from older jQuery event methods, see `.bind()`, `.delegate()`, and `.live()`. To remove events bound with `.on()`, see `.off()`. To attach an event that runs only once and then removes itself, see `.one()`.

## Event names and namespaces

Any event names can be used for the `events` argument. jQuery will pass through the browser's standard JavaScript event types, calling the `handler` function when the browser generates events due to user actions such as `click`. In addition, the `.trigger()` method can trigger both standard browser event names and custom event names to call attached handlers.

An event name can be qualified by *event namespaces* that simplify removing or triggering the event. For example,

`"click.myPlugin.simple"` defines both the `myPlugin` and `simple` namespaces for this particular click event. A click event handler attached via that string could be removed with

`.off("click.myPlugin")` OR `.off("click.simple")` without disturbing other click handlers attached to the elements. Namespaces are similar to CSS classes in that they are not hierarchical; only one name needs to match. Namespaces beginning with an underscore are reserved for jQuery's use.

In the second form of `.on()`, the `events` argument is a plain object. The keys are strings in the same form as the `events` argument with space-separated event type names and optional namespaces. The value for each key is a function (or `false` value) that is used as the `handler` instead of the final argument to the method. In other respects, the two forms are identical in their behavior as described below.

## Direct and delegated events

The majority of browser events *bubble*, or *propagate*, from the deepest, innermost element (the **event target**) in the document where they occur all the way up to the body and the `document` element. In Internet Explorer 8 and lower, a few events such as `change` and `submit` do not natively bubble but jQuery patches these to bubble and create consistent cross-browser behavior.

If `selector` is omitted or is null, the event handler is referred to as *direct* or *directly-bound*. The handler is called every time an event occurs on the selected elements, whether it occurs directly on the element or bubbles from a descendant (inner) element.

When a `selector` is provided, the event handler is referred to as *delegated*. The handler is not called when the event occurs directly on the bound element, but only for descendants (inner elements) that match the selector. jQuery bubbles the event from the event target up to the element where the handler is attached (i.e., innermost to outermost element) and runs the handler for any elements along that path matching the selector.

**Event handlers are bound only to the currently selected elements; they must exist on the page at the time your code makes the call to `.on()`.** To ensure the elements are present and can be selected, perform event binding inside a document ready handler for elements that are in the HTML markup on the page. If new HTML is being injected into the page, select the elements and attach event handlers *after* the new HTML is placed into the page. Or, use delegated events to attach an event handler, as described next.

Delegated events have the advantage that they can process events from *descendant elements* that are added to the document at a later time. By picking an element that is guaranteed to be present at the time the delegated event handler is attached, you can use delegated events to avoid the need to frequently attach and remove event handlers. This element could be the container element of a view in a Model-View-Controller design, for example, or `document` if the event handler wants to monitor all bubbling events in the document. The

`document` element is available in the `head` of the document before loading any other HTML, so it is safe to attach events there without waiting for the document to be ready.

In addition to their ability to handle events on descendant elements not yet created, another advantage of delegated events is their potential for much lower overhead when many elements must be monitored. On a data table with 1,000 rows in its `tbody`, this example attaches a handler to 1,000 elements:

```
1 | $("#dataTable tbody tr").on("click", function()
2 |     alert($(this).text());
3 | });
```

A delegated-events approach attaches an event handler to only one element, the `tbody`, and the event only needs to bubble up one level (from the clicked `tr` to `tbody`):

```
1 | $("#dataTable tbody").on("click", "tr", funct:
2 |     alert($(this).text());
3 | });
```

**Note:** Delegated events do not work for SVG.

## The event handler and its environment

The `handler` argument is a function (or the value `false`, see below), and is required unless you pass an object for the `events` argument. You can provide an anonymous handler function at the point of the `.on()` call, as the examples have done above, or declare a named function and pass its name:

```
1 | function notify() { alert("clicked"); }
2 | $("button").on("click", notify);
```

When the browser triggers an event or other JavaScript calls jQuery's `.trigger()` method, jQuery passes the handler an `event object` it can use to analyze and change the status of the event. This object is a *normalized subset* of data provided by the browser; the browser's unmodified native event object is available in `event.originalEvent`. For example, `event.type` contains the event name (e.g., "resize") and `event.target` indicates the deepest (innermost) element where the event occurred.

By default, most events bubble up from the original event target to the `document` element. At each element along the way, jQuery calls any matching event handlers that have been attached. A handler can prevent the event from bubbling further up the document tree (and thus prevent handlers on those elements from running) by calling `event.stopPropagation()`. Any other handlers attached on the current element *will* run however. To prevent that, call `event.stopImmediatePropagation()`. (Event handlers bound to an element are called in the same order that they were bound.)

Similarly, a handler can call `event.preventDefault()` to cancel any default action that the browser may have for this event; for example, the default action on a `click` event is to follow the link. Not all browser events have default actions, and not all default actions can be canceled. See the [W3C Events Specification](#) for details.

Returning `false` from an event handler will automatically call `event.stopPropagation()` and `event.preventDefault()`. A `false` value

can also be passed for the `handler` as a shorthand for `function(){ return false; }`. So, `$(".disabled").on("click", false);` attaches an event handler to all links with class "disabled" that prevents them from being followed when they are clicked and also stops the event from bubbling.

When jQuery calls a handler, the `this` keyword is a reference to the element where the event is being delivered; for directly bound events this is the element where the event was attached and for delegated events this is an element matching `selector`. (Note that `this` may not be equal to `event.target` if the event has bubbled from a descendant element.) To create a jQuery object from the element so that it can be used with jQuery methods, use `$(this)`.

## Passing data to the handler

If a `data` argument is provided to `.on()` and is not `null` or `undefined`, it is passed to the handler in the `event.data` property each time an event is triggered. The `data` argument can be any type, but if a string is used the `selector` must either be provided or explicitly passed as `null` so that the data is not mistaken for a selector. Best practice is to use a plain object so that multiple values can be passed as properties.

As of jQuery 1.4, the same event handler can be bound to an element multiple times. This is especially useful when the `event.data` feature is being used, or when other unique data resides in a closure around the event handler function. For example:

```
1 | function greet(event) { alert("Hello "+event.data.name); }
2 | $("button").on("click", { name: "Karl" }, greet);
3 | $("button").on("click", { name: "Addy" }, greet);
```

The above code will generate two different alerts when the button is clicked.

As an alternative or in addition to the `data` argument provided to the `.on()` method, you can also pass data to an event handler using a second argument to `.trigger()` or `.triggerHandler()`.

## Event performance

In most cases, an event such as `click` occurs infrequently and performance is not a significant concern. However, high frequency events such as `mousemove` or `scroll` can fire dozens of times per second, and in those cases it becomes more important to use events judiciously. Performance can be increased by reducing the amount of work done in the handler itself, caching information needed by the handler rather than recalculating it, or by rate-limiting the number of actual page updates using `setTimeout`.

Attaching many delegated event handlers near the top of the document tree can degrade performance. Each time the event occurs, jQuery must compare all selectors of all attached events of that type to every element in the path from the event target up to the top of the document. For best performance, attach delegated events at a document location as close as possible to the target elements. Avoid excessive use of `document` or `document.body` for delegated events on large documents.

jQuery can process simple selectors of the form `tag#id.class` very quickly when they are used to filter delegated events. So, `"#myForm"`, `"a.external"`, and `"button"` are all fast selectors. Delegated events that use more complex selectors, particularly hierarchical ones, can be several times slower--although they are still fast enough for most applications. Hierarchical selectors can often be avoided simply by attaching the handler to a more appropriate point in the document. For example, instead of `$("#body").on("click", "#commentForm .addNew", addComment)` USE `$("#commentForm").on("click", ".addNew", addComment)`.

## Additional notes

There are shorthand methods for some events such as `.click()` that can be used to attach or trigger event handlers. For a complete list of shorthand methods, see the [events category](#).

*Deprecated as of jQuery 1.8:* The name `"hover"` used as a shorthand for the string `"mouseenter mouseleave"`. It attaches a *single event handler* for those two events, and the handler must examine `event.type` to determine whether the event is `mouseenter` or `mouseleave`. Do not confuse the "hover" pseudo-event-name with the `.hover()` method, which accepts *one or two* functions.

jQuery's event system requires that a DOM element allow attaching data via a property on the element, so that events can be tracked and delivered. The `object`, `embed`, and `applet` elements cannot attach data, and therefore cannot have jQuery events bound to them.

The `focus` and `blur` events are specified by the W3C to not bubble, but jQuery defines cross-browser `focusin` and `focusout` events that do bubble. When `focus` and `blur` are used to attach delegated event handlers, jQuery maps the names and delivers them as `focusin` and `focusout` respectively. For consistency and clarity, use the bubbling event type names.

In all browsers, the `load`, `scroll`, and `error` events (e.g., on an `<img>` element) do not bubble. In Internet Explorer 8 and lower, the `paste` and `reset` events do not bubble. Such events are not supported for use with delegation, but they *can* be used when the event handler is directly attached to the element generating the event.

The `error` event on the `window` object uses nonstandard arguments and return value conventions, so it is not supported by jQuery. Instead, assign a handler function directly to the `window.onerror` property.

## Examples:

**Example:** *Display a paragraph's text in an alert when it is clicked:*

```
1 | $("p").on("click", function(){
2 | alert( $(this).text() );
3 | });
```

**Example:** *Pass data to the event handler, which is specified here by name:*

```
1 | function myHandler(event) {
2 |     alert(event.data.foo);
3 | }
4 | $("p").on("click", {foo: "bar"}, myHandler)
```

**Example:** *Cancel a form submit action and prevent the event from bubbling up by returning `false`:*

```
1 | $("form").on("submit", false)
```

**Example:** *Cancel only the default action by using `.preventDefault()`.*

```
1 | $("form").on("submit", function(event) {
2 |     event.preventDefault();
3 | });
```

**Example:** *Stop submit events from bubbling without*

## preventing form submit, using `.stopPropagation()`.

```
1 | $("form").on("submit", function(event) {
2 |     event.stopPropagation();
3 | });
```

## Example: Attach and trigger custom (non-browser) events.

```
1 | <!DOCTYPE html>
2 | <html>
3 | <head>
4 |     <style>
5 |     p { color:red; }
6 |     span { color:blue; }
7 |     </style>
8 |     <script src="http://code.jquery.com/jquery-1.10.2.min.js"></script>
9 | </head>
10 | <body>
11 |     <p>Has an attached custom event.</p>
12 |     <button>Trigger custom event</button>
13 |     <span style="display:none;"></span>
14 |     <script>
15 |     $("p").on("myCustomEvent", function(e, myName) {
16 |         $(this).text(myName + ", hi there!");
17 |         $("span").stop().css("opacity", 1)
18 |             .text("myName = " + myName)
19 |             .fadeIn(30).fadeOut(1000);
20 |     });
21 |     $("button").click(function () {
22 |         $("p").trigger("myCustomEvent", [ "John" ]);
23 |     });
24 | </script>
25 |
26 | </body>
```

27 | </html>

Demo

**Example** · *Attach multiple event handlers simultaneously using a plain object.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5    .test { color: #000; padding: .5em; border: 1
6    .active { color: #900;}
7    .inside { background-color: aqua; }
8  </style>
9    <script src="http://code.jquery.com/jquery-
10 </head>
11 <body>
12   <div class="test">test div</div>
13   <script>$("div.test").on({
14     click: function(){
15       $(this).toggleClass("active");
16     },
17     mouseenter: function(){
18       $(this).addClass("inside");
19     },
20     mouseleave: function(){
21       $(this).removeClass("inside");
22     }
23   });</script>
24
25 </body>
26 </html>
```

Demo

**Example:** Click any paragraph to add another after it. Note that `on()` allows a click event on any paragraph--even new ones--since the event is handled by the ever-present body element after it bubbles to there.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5   p { background:yellow; font-weight:bold; cursor:
6     padding:5px; }
7   p.over { background: #ccc; }
8   span { color:red; }
9 </style>
10  <script src="http://code.jquery.com/jquery-
11 </head>
12 <body>
13   <p>Click me!</p>
14   <span></span>
15 <script>
16 var count = 0;
17 $("body").on("click", "p", function(){
18   $(this).after("<p>Another paragraph! "+(++c
19 });
20 </script>
21
22 </body>
23 </html>
```

Demo

**Example:** Display each paragraph's text in an alert box whenever it is clicked:

```
1 $("body").on("click", "p", function(){
```

```
2 | alert( $(this).text() );  
3 | });
```

**Example:** *Cancel a link's default action using the `preventDefault` method.*

```
1 | $("body").on("click", "a", function(event){  
2 |     event.preventDefault();  
3 | });
```

POWERED BY **HERONOTE**

A new version of this book is available!

---

# .one()

Categories: [Events](#) > [Event Handler Attachment](#)

`.one( events [, data ], handler(eventObject) )`

Returns: [jQuery](#)

**Description:** *Attach a handler to an event for the elements. The handler is executed at most once per element.*

`.one( events [, data ],  
handler(eventObject) )`

version added: [1.1](#)

**events**

Type: [String](#)

A string containing one or more JavaScript event types, such as "click" or "submit," or custom event names.

**data**

Type: [PlainObject](#)

An object containing data that will be passed to the event handler.

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute at the time the event is triggered.

`.one( events [, selector ] [, data ],  
handler(eventObject) )`

version added: [1.7](#)

**events**

Type: [String](#)

One or more space-separated event types and optional namespaces, such as "click" or "keydown.myPlugin".

**selector**

Type: [String](#)

A selector string to filter the descendants of the selected elements that trigger the event. If the selector is `null` or omitted, the event is always triggered when it reaches the selected element.

**data**

Type: [Anything](#)

Data to be passed to the handler in `event.data` when an event is triggered.

### **handler(eventObject)**

Type: [Function\(\)](#)

A function to execute when the event is triggered. The value `false` is also allowed as a shorthand for a function that simply does `return false`.

## **.one( events [, selector ] [, data ] )**

version added: [1.7](#)

### **events**

Type: [PlainObject](#)

An object in which the string keys represent one or more space-separated event types and optional namespaces, and the values represent a handler function to be called for the event(s).

### **selector**

Type: [String](#)

A selector string to filter the descendants of the selected elements that will call the handler. If the selector is null or omitted, the handler is always called when it reaches the selected element.

### **data**

Type: [Anything](#)

Data to be passed to the handler in `event.data` when an event occurs.

The first form of this method is identical to `.bind()`, except that the handler is unbound after its first invocation. The second two forms, introduced in jQuery 1.7, are identical to `.on()` except that the handler is removed after the first time the event occurs at the delegated element, whether the selector matched anything or not. For example:

```
1 | $("#foo").one("click", function() {
2 |     alert("This will be displayed only once.");
3 | });
4 | $("body").one("click", "#foo", function() {
```

```
5 | alert("This displays if #foo is the first thing clicked");
6 | });
```

After the code is executed, a click on the element with ID `foo` will display the alert. Subsequent clicks will do nothing. This code is equivalent to:

```
1 | $("#foo").on("click", function( event ) {
2 |     alert("This will be displayed only once.");
3 |     $(this).off( event );
4 | });
```

In other words, explicitly calling `.off()` from within a regularly-bound handler has exactly the same effect.

If the first argument contains more than one space-separated event types, the event handler is called *once for each event type*.

## Examples:

**Example:** *Tie a one-time click to each div.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5   div { width:60px; height:60px; margin:5px; float:left;
6   background:green; border:10px outset;
7   cursor:pointer; }
8   p { color:red; margin:0; clear:left; }
9 </style>
10  <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
11 </head>
12 <body>
13
14 <div></div>
15 <div></div>
16 <div></div>
17 <div></div>
18 <div></div>
19
20 <p>Click a green square...</p>
21
22 <script>
23 var n = 0;
24 $("div").one("click", function() {
25   var index = $("div").index(this);
26   $(this).css({
27     borderStyle:"inset",
28     cursor:"auto"
29   });
30   $("p").text("Div at index #" + index + " clicked. Total clicks: " +
31     n + " That's " + ++n + " total clicks.");
```

```
32 | });  
33 |  
34 | </script>  
35 |  
36 | </body>  
37 | </html>
```

### Demo

**Example:** *To display the text of all paragraphs in an alert box the first time each of them is clicked:*

```
1 | $("p").one("click", function(){  
2 | alert( $(this).text() );  
3 | });
```

POWERED BYHERONOTE

A new version of this book is available!

---

# :only-child Selector

Categories: [Selectors](#) > [Child Filter](#)

## only-child selector

**Description:** *Selects all elements that are the only child of their parent.*

**jQuery( ":only-child" )**

version added: [1.1.4](#)

If the parent has other child elements, nothing is matched.

## Example:

*Change the text and add a border for each button that is the only child of its parent.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5      div { width:100px; height:80px; margin:5px;
6    </style>
7    <script src="http://code.jquery.com/jquery-
8  </head>
9  <body>
10   <div>
11     <button>Sibling! </button>
12     <button>Sibling! </button>
13   </div>
14
15   <div>
16     <button>Sibling! </button>
17   </div>
18   <div>
19     None
20   </div>
21
22   <div>
23     <button>Sibling! </button>
24     <button>Sibling! </button>
25     <button>Sibling! </button>
26   </div>
27
28   <div>
29     <button>Sibling! </button>
30   </div>
```

```
31 <script>
32   $("div button:only-child").text("Alone").css(
33 </script>
34
35 </body>
36 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# :only-of-type Selector

Categories: [Selectors](#) > [Child Filter](#)

## only-of-type selector

**Description:** *Selects all elements that have no siblings with the same element name.*

**jQuery( ":only-of-type" )**

version added: [1.9](#)

If the parent has other child elements with the same element name, nothing is matched.

## Example:

*Change the text and add a border for each button that is the only child button of its parent.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     div { width:100px; height:80px; margin:5px;
6     span { padding: 2px; margin: 3px; line-heig
7 </style>
8   <script src="http://code.jquery.com/jquery-
9 </head>
10 <body>
11   <div>
12     <button>Sibling! </button>
13     <button>Sibling! </button>
14   </div>
15
16   <div>
17     <button>Sibling! </button>
18   </div>
19   <div>
20     None
21   </div>
22
23   <div>
24     <button>Sibling! </button>
25     <span>Sibling! </span>
26     <span>Sibling! </span>
27
28   </div>
29   <div>
30     <button>Sibling! </button>
```

```
31 </div>
32 <script>
33     $("button:only-of-type").text("Alone").css("border"
34 </script>
35
36 </body>
37 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# .outerHeight()

Categories: [CSS](#) | [Dimensions](#) | [Manipulation](#) > [Style Properties](#)

`.outerHeight( [includeMargin ] )`

Returns: [Integer](#)

**Description:** *Get the current computed height for the first element in the set of matched elements, including padding, border, and optionally margin. Returns an integer (without "px") representation of the value or null if called on an empty set of elements.*

`.outerHeight( [includeMargin ] )`

version added: [1.2.6](#)

**includeMargin**

Type: [Boolean](#)

A Boolean indicating whether to include the element's margin in the calculation.

The top and bottom padding and border are always included in the `.outerHeight()` calculation; if the `includeMargin` argument is set to `true`, the margin (top and bottom) is also included.

This method is not applicable to `window` and `document` objects; for these, use `.height()` instead.



## Example:

*Get the outerHeight of a paragraph.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>p { margin:10px;padding:5px;border:2px solid red;}
5    <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
6  </head>
7  <body>
8    <p>Hello</p><p></p>
9    <script>var p = $("p:first");
10   $("p:last").text( "outerHeight:" + p.outerHeight() );
11
12  </body>
13  </html>
```

Demo

POWERED BY **HERONOTE**

A new version of this book is available!

---

# .outerWidth()

Categories: [CSS](#) | [Dimensions](#) | [Manipulation](#) > [Style Properties](#)

`.outerWidth( [includeMargin ] )`

Returns: [Integer](#)

**Description:** *Get the current computed width for the first element in the set of matched elements, including padding and border.*

`.outerWidth( [includeMargin ] )`

version added: [1.2.6](#)

**includeMargin**

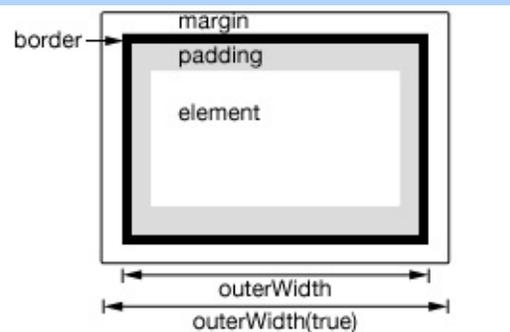
Type: [Boolean](#)

A Boolean indicating whether to include the element's margin in the calculation.

Returns the width of the element, along with left and right padding, border, and optionally margin, in pixels.

If `includeMargin` is omitted or `false`, the padding and border are included in the calculation; if `true`, the margin is also included.

This method is not applicable to `window` and `document` objects; for these, use `.width()` instead.



## Example:

*Get the `outerWidth` of a paragraph.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5      p { margin:10px;padding:5px;border:2px solid
6    </style>
7      <script src="http://code.jquery.com/jquery-
8    </head>
9    <body>
10     <p>Hello</p><p></p>
11     <script>var p = $("p:first");
12     $("p:last").text( "outerWidth:" + p.outerWidth
13   </script>
14
15   </body>
16   </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# .parent()

Categories: [Traversing](#) > [Tree Traversal](#)

`.parent( [selector ] )`

Returns: [jQuery](#)

**Description:** *Get the parent of each element in the current set of matched elements, optionally filtered by a selector.*

`.parent( [selector ] )`

version added: [1.0](#)

**selector**

Type: [Selector](#)

A string containing a selector expression to match elements against.

Given a jQuery object that represents a set of DOM elements, the `.parent()` method allows us to search through the parents of these elements in the DOM tree and construct a new jQuery object from the matching elements. The `.parents()` and `.parent()` methods are similar, except that the latter only travels a single level up the DOM tree.

The method optionally accepts a selector expression of the same type that we can pass to the `$( )` function. If the selector is supplied, the elements will be filtered by testing whether they match it.

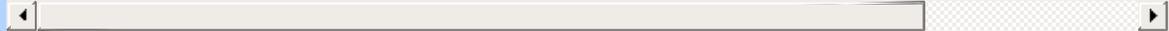
Consider a page with a basic nested list on it:

```
1 <ul class="level-1">
2   <li class="item-i">I</li>
3   <li class="item-ii">II
4     <ul class="level-2">
5       <li class="item-a">A</li>
6       <li class="item-b">B
7         <ul class="level-3">
8           <li class="item-1">1</li>
9           <li class="item-2">2</li>
10          <li class="item-3">3</li>
11        </ul>
```

```
12     </li>
13     <li class="item-c">C</li>
14   </ul>
15 </li>
16 <li class="item-iii">III</li>
17 </ul>
```

If we begin at item A, we can find its parents:

```
1 | $('li.item-a').parent().css('background-color
```



The result of this call is a red background for the level-2 list. Since we do not supply a selector expression, the parent element is unequivocally included as part of the object. If we had supplied one, the element would be tested for a match before it was included.

## Examples:

**Example:** Shows the parent of each element as (parent > child). Check the View Source to see the raw html.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5   div,p { margin:10px; }
6   </style>
7   <script src="http://code.jquery.com/jquery-
8 </head>
9 <body>
10  <div>div,
11    <span>span, </span>
12    <b>b </b>
13
14  </div>
15  <p>p,
16    <span>span,
17    <em>em </em>
18    </span>
19  </p>
20
21  <div>div,
22    <strong>strong,
23    <span>span, </span>
24    <em>em,
25    <b>b, </b>
26    </em>
27
28    </strong>
29    <b>b </b>
30  </div>
```

```
31 <script>
32 $( "*", document.body).each(function () {
33     var parentTag = $(this).parent().get(0).tagName;
34     $(this).prepend(document.createTextNode(parentTag +
35 });
36 </script>
37
38 </body>
39 </html>
```

Demo

**Example:** Find the parent element of each paragraph with a class "selected".

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <script src="http://code.jquery.com/jquery-
5 </head>
6 <body>
7     <div><p>Hello</p></div>
8     <div class="selected"><p>Hello Again</p></d.
9
10 <script>$("p").parent(".selected").css("backg
11
12 </body>
13 </html>
```

Demo

**A new version of this book is available!**

---

# :parent Selector

Categories: [Selectors](#) > [Content Filter](#) | [Selectors](#) > [jQuery Extensions](#)

## parent selector

**Description:** *Select all elements that have at least one child node (either an element or text).*

jQuery( ":parent" )

version added: [1.0](#)

This is the inverse of `:empty`.

One important thing to note regarding the use of `:parent` (and `:empty`) is that child nodes include text nodes.

The W3C recommends that the `<p>` element have at least one child node, even if that child is merely text (see <http://www.w3.org/TR/html401/struct/text.html#edef-P>). Some other elements, on the other hand, are empty (i.e. have no children) by definition: `<input>`, `<img>`, `<br>`, and `<hr>`, for example.

### Additional Notes:

Because `:parent` is a jQuery extension and not part of the CSS specification, queries using `:parent` cannot take advantage of the performance boost provided by the native DOM `querySelectorAll()` method. To achieve the best performance when using `:parent` to select elements, first select the elements using a pure CSS selector, then use `.filter(":parent")`.

## Example:

*Finds all tds with children, including text.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     td { width:40px; background:green; }
6   </style>
7   <script src="http://code.jquery.com/jquery-
8 </head>
9 <body>
10  <table border="1">
11    <tr><td>Value 1</td><td></td></tr>
12    <tr><td>Value 2</td><td></td></tr>
13  </table>
14  <script>$("td:parent").fadeTo(1500, 0.3);</sc
15
16 </body>
17 </html>
```

Demo

POWERED BY HERONOTE

A new version of this book is available!

---

# .parents()

Categories: [Traversing](#) > [Tree Traversal](#)

`.parents( [selector ] )`

Returns: [jQuery](#)

**Description:** *Get the ancestors of each element in the current set of matched elements, optionally filtered by a selector.*

`.parents( [selector ] )`

version added: [1.0](#)

**selector**

Type: [Selector](#)

A string containing a selector expression to match elements against.

Given a jQuery object that represents a set of DOM elements, the `.parents()` method allows us to search through the ancestors of these elements in the DOM tree and construct a new jQuery object from the matching elements ordered from immediate parent on up; the elements are returned in order from the closest parent to the outer ones. The `.parents()` and `.parent()` methods are similar, except that the latter only travels a single level up the DOM tree.

The method optionally accepts a selector expression of the same type that we can pass to the `$( )` function. If the selector is supplied, the elements will be filtered by testing whether they match it.

Consider a page with a basic nested list on it:

```
1 | <ul class="level-1">
2 |   <li class="item-i">I</li>
3 |   <li class="item-ii">II
4 |     <ul class="level-2">
5 |       <li class="item-a">A</li>
6 |       <li class="item-b">B
7 |         <ul class="level-3">
8 |           <li class="item-1">1</li>
9 |           <li class="item-2">2</li>
10 |          <li class="item-3">3</li>
```

```
11 |         </ul>
12 |     </li>
13 |     <li class="item-c">C</li>
14 | </ul>
15 | </li>
16 | <li class="item-iii">III</li>
17 | </ul>
```

If we begin at item A, we can find its ancestors:

```
1 | $('li.item-a').parents().css('background-color');
```

The result of this call is a red background for the level-2 list, item II, and the level-1 list (and on up the DOM tree all the way to the `<html>` element). Since we do not supply a selector expression, all of the ancestors are part of the returned jQuery object. If we had supplied one, only the matching items among these would be included.

## Examples:

**Example:** *Find all parent elements of each b.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5      b, span, p, html body {
6        padding: .5em;
7        border: 1px solid;
8      }
9      b { color:blue; }
10     strong { color:red; }
11   </style>
12   <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
13 </head>
14 <body>
15   <div>
16     <p>
17       <span>
18         <b>My parents are: </b>
19       </span>
20     </p>
21   </div>
22   <script>
23     var parentEls = $("b").parents()
24       .map(function () {
25         return this.tagName;
26       })
27       .get().join(", ");
28     $("b").append("<strong>" + parentEls + "</strong>");
29   </script>
30
31 </body>
```

32 | </html>

Demo

**Example** · Click to find all unique div parent elements of each span.

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5
6    p, div, span {margin:2px; padding:1px; }
7    div { border:2px white solid; }
8    span { cursor:pointer; font-size:12px; }
9    .selected { color:blue; }
10   b { color:red; display:block; font-size:14px; }
11 </style>
12 <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
13 </head>
14 <body>
15   <p>
16     <div>
17       <div><span>Hello</span></div>
18       <span>Hello Again</span>
19
20     </div>
21     <div>
22       <span>And Hello Again</span>
23     </div>
24   </p>
25
26   <b>Click Hellos to toggle their parents.</b>
27 <script>
28 function showParents() {
29   $("div").css("border-color", "white");
```

```
30     var len = $("span.selected")
31             .parents("div")
32             .css("border", "2px red solid")
33             .length;
34     $("b").text("Unique div parents: " + len);
35 }
36 $("span").click(function () {
37     $(this).toggleClass("selected");
38     showParents();
39 });</script>
40
41 </body>
42 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# .parentsUntil()

Categories: [Traversing](#) > [Tree Traversal](#)

`.parentsUntil( [selector ] [, filter ] )`

Returns: [jQuery](#)

**Description:** *Get the ancestors of each element in the current set of matched elements, up to but not including the element matched by the selector, DOM node, or jQuery object.*

`.parentsUntil( [selector ] [, filter ] )`

version added: [1.4](#)

**selector**

Type: [Selector](#)

A string containing a selector expression to indicate where to stop matching ancestor elements.

**filter**

Type: [Selector](#)

A string containing a selector expression to match elements against.

`.parentsUntil( [element ] [, filter ] )`

version added: [1.6](#)

**element**

Type: [Element](#)

A DOM node or jQuery object indicating where to stop matching ancestor elements.

**filter**

Type: [Selector](#)

A string containing a selector expression to match elements against.

Given a selector expression that represents a set of DOM elements, the `.parentsUntil()` method traverses through the ancestors of these elements until it reaches an element matched by the selector passed in the method's argument. The resulting jQuery object contains all of the ancestors up to but not including the one matched by the `.parentsUntil()` selector.

If the selector is not matched or is not supplied, all ancestors will be selected; in these cases it selects the same elements as the

`.parents()` method does when no selector is provided.

**As of jQuery 1.6**, A DOM node or jQuery object, instead of a selector, may be used for the first `.parentsUntil()` argument.

The method optionally accepts a selector expression for its second argument. If this argument is supplied, the elements will be filtered by testing whether they match it.

## Example:

Find the ancestors of `<li class="item-a">` up to `<ul class="level-1">` and give them a red background color. Also find ancestors of `<li class="item-2">` that have a class of "yes" up to `<ul class="level-1">` and give them a green border.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-
5 </head>
6 <body>
7
8 <ul class="level-1 yes">
9   <li class="item-i">I</li>
10  <li class="item-ii">II
11    <ul class="level-2 yes">
12      <li class="item-a">A</li>
13      <li class="item-b">B
14        <ul class="level-3">
15          <li class="item-1">1</li>
16          <li class="item-2">2</li>
17          <li class="item-3">3</li>
18        </ul>
19      </li>
20      <li class="item-c">C</li>
21    </ul>
22  </li>
23  <li class="item-iii">III</li>
24 </ul>
25 <script>
26 $( "li.item-a" ).parentsUntil( ".level-1" )
27   .css( "background-color", "red" );
```

```
28
29 $( "li.item-2").parentsUntil( $( "ul.level-1"), ".yes"
30   .css( "border", "3px solid green");
31
32 </script>
33
34 </body>
35 </html>
```

Demo

POWERED BYHERONOTE

A new version of this book is available!

---

# :password Selector

Categories: [Selectors](#) > [Form](#) | [Selectors](#) > [jQuery Extensions](#)

## password selector

**Description:** *Selects all elements of type password.*

**jQuery( ":password" )**

version added: [1.0](#)

`$( ':password' )` is equivalent to `$( '[type=password]' )`. As with other pseudo-class selectors (those that begin with a ":" ) it is recommended to precede it with a tag name or some other selector; otherwise, the universal selector ("\*") is implied. In other words, the bare `$( ':password' )` is equivalent to `$( '*:password' )`, so `$( 'input:password' )` should be used instead.

### Additional Notes:

Because `:password` is a jQuery extension and not part of the CSS specification, queries using `:password` cannot take advantage of the performance boost provided by the native DOM `querySelectorAll()` method. For better performance in modern browsers, use `[type="password"]` instead.

## Example:

*Finds all password inputs.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5      textarea { height:45px; }
6    </style>
7    <script src="http://code.jquery.com/jquery-
8  </head>
9  <body>
10   <form>
11     <input type="button" value="Input Button"
12     <input type="checkbox" />
13
14     <input type="file" />
15     <input type="hidden" />
16     <input type="image" />
17
18     <input type="password" />
19     <input type="radio" />
20     <input type="reset" />
21
22     <input type="submit" />
23     <input type="text" />
24     <select><option>Option</option></select>
25
26     <textarea></textarea>
27     <button>Button</button>
28   </form>
29   <div>
30   </div>
31 </script>
```

```
32 var input = $("input:password").css({background:
33 $("div").text("For this type jQuery found " + input.length
34 .css("color", "red");
35 $("form").submit(function () { return false; });
36 </script>
37
38 </body>
39 </html>
```

Demo

POWERED BYHERONOTE

A new version of this book is available!

---

# .position()

Categories: [CSS](#) | [Offset](#) | [Manipulation](#) > [Style Properties](#)

`.position()`

Returns: `Object`

**Description:** *Get the current coordinates of the first element in the set of matched elements, relative to the offset parent.*

`.position()`

version added: `1.2`

This method does not accept any arguments.

The `.position()` method allows us to retrieve the current position of an element *relative to the offset parent*. Contrast this with `.offset()`, which retrieves the current position *relative to the document*. When positioning a new element near another one and within the same containing DOM element, `.position()` is the more useful.

Returns an object containing the properties `top` and `left`.

**Note:** *jQuery does not support getting the position coordinates of hidden elements or accounting for borders, margins, or padding set on the body element.*

## Example:

*Access the position of the second paragraph:*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5
6    div { padding: 15px;}
7    p { margin-left:10px; }
8  </style>
9    <script src="http://code.jquery.com/jquery-
10 </head>
11 <body>
12
13 <div>
14   <p>Hello</p>
15 </div>
16 <p></p>
17
18 <script>
19 var p = $("p:first");
20 var position = p.position();
21 $("p:last").text( "left: " + position.left +
22 </script>
23
24 </body>
25 </html>
```

Demo

**A new version of this book is available!**

---

# .prepend()

Categories: [Manipulation](#) > [DOM Insertion, Inside](#)

`.prepend( content [, content ] )`

Returns: [jQuery](#)

**Description:** *Insert content, specified by the parameter, to the beginning of each element in the set of matched elements.*

`.prepend( content [, content ] )`

version added: [1.0](#)

**content**

Type: [htmlString](#) or [Element](#) or [jQuery](#)

DOM element, array of elements, HTML string, or jQuery object to insert at the beginning of each element in the set of matched elements.

**content**

Type: [htmlString](#) or [Element](#) or [jQuery](#)

One or more additional DOM elements, arrays of elements, HTML strings, or jQuery objects to insert at the beginning of each element in the set of matched elements.

`.prepend( function(index, html) )`

version added: [1.4](#)

**function(index, html)**

Type: [Function\(\)](#)

A function that returns an HTML string, DOM element(s), or jQuery object to insert at the beginning of each element in the set of matched elements. Receives the index position of the element in the set and the old HTML value of the element as arguments. Within the function, `this` refers to the current element in the set.

The `.prepend()` method inserts the specified content as the first child of each element in the jQuery collection (To insert it as the *last* child, use `.append()`).

The `.prepend()` and `.prependTo()` methods perform the same task. The major difference is in the syntax—specifically, in the placement of the content and target. With `.prepend()`, the selector expression preceding the method is the container into which the content is inserted. With `.prependTo()`, on the other hand, the content precedes

the method, either as a selector expression or as markup created on the fly, and it is inserted into the target container.

Consider the following HTML:

```
1 | <h2>Greetings</h2>
2 | <div class="container">
3 |   <div class="inner">Hello</div>
4 |   <div class="inner">Goodbye</div>
5 | </div>
```

You can create content and insert it into several elements at once:

```
1 | $(' .inner ').prepend( '<p>Test</p>' );
```

Each `<div class="inner">` element gets this new content:

```
1 | <h2>Greetings</h2>
2 | <div class="container">
3 |   <div class="inner">
4 |     <p>Test</p>
5 |     Hello
6 |   </div>
7 |   <div class="inner">
8 |     <p>Test</p>
9 |     Goodbye
10 |   </div>
11 | </div>
```

You can also select an element on the page and insert it into another:

```
1 | $(' .container ').prepend($(' h2 '));
```

If a *single element* selected this way is inserted into a single location elsewhere in the DOM, it will be moved into the target (*not cloned*):

```
1 | <div class="container">
2 |     <h2>Greetings</h2>
3 |     <div class="inner">Hello</div>
4 |     <div class="inner">Goodbye</div>
5 | </div>
```

**Important:** If there is more than one target element, however, cloned copies of the inserted element will be created for each target after the first.

### Additional Arguments

Similar to other content-adding methods such as `.append()` and `.before()`, `.prepend()` also supports passing in multiple arguments as input. Supported input includes DOM elements, jQuery objects, HTML strings, and arrays of DOM elements.

For example, the following will insert two new `<div>`s and an existing `<div>` as the first three child nodes of the body:

```
1 | var $newdiv1 = $('<div id="object1"/>'),
2 |     newdiv2 = document.createElement('div'),
3 |     existingdiv1 = document.getElementById('f
4 |
5 | $('body').prepend($newdiv1, [newdiv2, existing
```

Since `.prepend()` can accept any number of additional arguments, the same result can be achieved by passing in the three `<div>`s as three separate arguments, like so: `$('body').prepend($newdiv1, newdiv2, existingdiv1)`. The type and number of arguments will largely depend on how you collect the elements in your code.



## Examples:

**Example:** *Prepends some HTML to all paragraphs.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>p { background:yellow; }</style>
5   <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
6 </head>
7 <body>
8   <p>there, friend!</p>
9
10  <p>amigo!</p>
11  <script>$("p").prepend("<b>Hello </b>");</script>
12
13 </body>
14 </html>
```

Demo

**Example:** *Prepends a DOM Element to all paragraphs.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>p { background:yellow; }</style>
5   <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
6 </head>
7 <body>
8   <p>is what I'd say</p>
9   <p>is what I said</p>
10  <script>$("p").prepend(document.createTextNode("<b>Hello </b>"));</script>
11
```

```
12 | </body>
13 | </html>
```

Demo

**Example:** *Prepends a jQuery object (similar to an Array of DOM Elements) to all paragraphs.*

```
1 | <!DOCTYPE html>
2 | <html>
3 | <head>
4 |   <style>p { background:yellow; }</style>
5 |   <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
6 | </head>
7 | <body>
8 |   <p> is what was said.</p><b>Hello</b>
9 |   <script>$("p").prepend( $("b") );</script>
10 |
11 | </body>
12 | </html>
```

Demo

POWERED BY HERONOTE

A new version of this book is available!

---

# .prependTo()

Categories: [Manipulation](#) > [DOM Insertion, Inside](#)

`.prependTo( target )`

Returns: `jQuery`

**Description:** *Insert every element in the set of matched elements to the beginning of the target.*

`.prependTo( target )`

version added: `1.0`

**target**

Type: `Selector` or `htmlString` or `Element` or `jQuery`

A selector, element, HTML string, or jQuery object; the matched set of elements will be inserted at the beginning of the element(s) specified by this parameter.

The `.prepend()` and `.prependTo()` methods perform the same task. The major difference is in the syntax-specifically, in the placement of the content and target. With `.prepend()`, the selector expression preceding the method is the container into which the content is inserted. With `.prependTo()`, on the other hand, the content precedes the method, either as a selector expression or as markup created on the fly, and it is inserted into the target container.

Consider the following HTML:

```
1 | <h2>Greetings</h2>
2 | <div class="container">
3 |   <div class="inner">Hello</div>
4 |   <div class="inner">Goodbye</div>
5 | </div>
```

We can create content and insert it into several elements at once:

```
1 | $( '<p>Test</p>' ).prependTo( '.inner' );
```

Each inner `<div>` element gets this new content:

```
1 | <h2>Greetings</h2>
2 | <div class="container">
3 |   <div class="inner">
4 |     <p>Test</p>
5 |     Hello
6 |   </div>
7 |   <div class="inner">
8 |     <p>Test</p>
9 |     Goodbye
10 |   </div>
11 | </div>
```

We can also select an element on the page and insert it into another:

```
1 | $('h2').prependTo($('.container'));
```

If an element selected this way is inserted into a single location elsewhere in the DOM, it will be moved into the target (not cloned):

```
1 | <div class="container">
2 |   <h2>Greetings</h2>
3 |   <div class="inner">Hello</div>
4 |   <div class="inner">Goodbye</div>
5 | </div>
```

If there is more than one target element, however, cloned copies of the inserted element will be created for each target after the first.

## Example:

*Prepend all spans to the element with the ID "foo"  
(Check `.prepend()` documentation for more examples)*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>div { background:yellow; }</style>
5   <script src="http://code.jquery.com/jquery-
6 </head>
7 <body>
8   <div id="foo">FOO!</div>
9
10  <span>I have something to say... </span>
11 <script>
12 $("span").prependTo("#foo");
13 </script>
14
15 </body>
16 </html>
```

Demo

POWERED BYHERONOTE

A new version of this book is available!

---

# .prev()

Categories: [Traversing](#) > [Tree Traversal](#)

`.prev( [selector ] )`

Returns: [jQuery](#)

**Description:** *Get the immediately preceding sibling of each element in the set of matched elements, optionally filtered by a selector.*

`.prev( [selector ] )`

version added: [1.0](#)

**selector**

Type: [Selector](#)

A string containing a selector expression to match elements against.

Given a jQuery object that represents a set of DOM elements, the `.prev()` method searches for the predecessor of each of these elements in the DOM tree and constructs a new jQuery object from the matching elements.

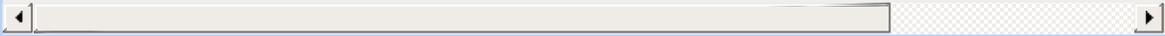
The method optionally accepts a selector expression of the same type that can be passed to the `$( )` function. If the selector is supplied, the preceding element will be filtered by testing whether it match the selector.

Consider a page with a simple list on it:

```
1 | <ul>
2 |     <li>list item 1</li>
3 |     <li>list item 2</li>
4 |     <li class="third-item">list item 3</li>
5 |     <li>list item 4</li>
6 |     <li>list item 5</li>
7 | </ul>
```

To select the element that comes immediately before item three:

```
1 | $( 'li.third-item' ).prev().css( 'background-color'
```



The result of this call is a red background behind item 2. Since no selector expression is supplied, this preceding element is unequivocally included as part of the object. If one had been supplied, the element would be tested for a match before it was included.

If no previous sibling exists, or if the previous sibling element does not match a supplied selector, an empty jQuery object is returned.

To select *all* preceding sibling elements, rather than just the preceding *adjacent* sibling, use the [.prevAll\(\)](#) method.

## Examples:

**Example:** *Find the very previous sibling of each div.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     div { width:40px; height:40px; margin:10px;
6           float:left; border:2px blue solid;
7           padding:2px; }
8     span { font-size:14px; }
9     p { clear:left; margin:10px; }
10  </style>
11  <script src="http://code.jquery.com/jquery-
12 </head>
13 <body>
14   <div></div>
15   <div></div>
16   <div><span>has child</span></div>
17
18   <div></div>
19   <div></div>
20   <div></div>
21   <div id="start"></div>
22
23   <div></div>
24   <p><button>Go to Prev</button></p>
25 <script>
26   var $curr = $("#start");
27   $curr.css("background", "#f99");
28   $("button").click(function () {
29     $curr = $curr.prev();
30     $("div").css("background", "");
31     $curr.css("background", "#f99");
```

```
32     });
33
34 </script>
35
36 </body>
37 </html>
```

Demo

**Example:** For each `naranranh`, find the very previous sibling that has a class "selected".

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-...
5 </head>
6 <body>
7   <div><span>Hello</span></div>
8
9   <p class="selected">Hello Again</p>
10  <p>And Again</p>
11  <script>$("p").prev(".selected").css("backgro
12
13 </body>
14 </html>
```

Demo

**A new version of this book is available!**

---

# .prevAll()

Categories: [Traversing](#) > [Tree Traversal](#)

`.prevAll( [selector ] )`

Returns: [jQuery](#)

**Description:** *Get all preceding siblings of each element in the set of matched elements, optionally filtered by a selector.*

`.prevAll( [selector ] )`

version added: [1.2](#)

**selector**

Type: [Selector](#)

A string containing a selector expression to match elements against.

Given a jQuery object that represents a set of DOM elements, the `.prevAll()` method searches through the predecessors of these elements in the DOM tree and construct a new jQuery object from the matching elements; the elements are returned in order beginning with the closest sibling.

The method optionally accepts a selector expression of the same type that we can pass to the `$( )` function. If the selector is supplied, the elements will be filtered by testing whether they match it.

Consider a page with a simple list on it:

```
1 | <ul>
2 |     <li>list item 1</li>
3 |     <li>list item 2</li>
4 |     <li class="third-item">list item 3</li>
5 |     <li>list item 4</li>
6 |     <li>list item 5</li>
7 | </ul>
```

If we begin at the third item, we can find the elements which come before it:

```
1 | $('li.third-item').prevAll().css('background-color')
```

The result of this call is a red background behind items 1 and 2. Since we do not supply a selector expression, these preceding elements are unequivocally included as part of the object. If we had supplied one, the elements would be tested for a match before they were included.

## Example:

*Locate all the divs preceding the last div and give them a class.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5
6   div { width:70px; height:70px; background:#
7         border:2px solid black; margin:10px;
8   div.before { border-color: red; }
9   </style>
10  <script src="http://code.jquery.com/jquery-
11 </head>
12 <body>
13   <div></div>
14   <div></div>
15   <div></div>
16   <div></div>
17 <script>$("div:last").prevAll().addClass("bef
18
19 </body>
20 </html>
```

Demo

**A new version of this book is available!**

---

# .prevUntil()

Categories: [Traversing](#) > [Tree Traversal](#)

`.prevUntil( [selector ] [, filter ] )`

Returns: [jQuery](#)

**Description:** *Get all preceding siblings of each element up to but not including the element matched by the selector, DOM node, or jQuery object.*

`.prevUntil( [selector ] [, filter ] )`

version added: [1.4](#)

**selector**

Type: [Selector](#)

A string containing a selector expression to indicate where to stop matching preceding sibling elements.

**filter**

Type: [Selector](#)

A string containing a selector expression to match elements against.

`.prevUntil( [element ] [, filter ] )`

version added: [1.6](#)

**element**

Type: [Element](#)

A DOM node or jQuery object indicating where to stop matching preceding sibling elements.

**filter**

Type: [Selector](#)

A string containing a selector expression to match elements against.

Given a selector expression that represents a set of DOM elements, the `.prevUntil()` method searches through the predecessors of these elements in the DOM tree, stopping when it reaches an element matched by the method's argument. The new jQuery object that is returned contains all previous siblings up to but not including the one matched by the `.prevUntil()` selector; the elements are returned in order from the closest sibling to the farthest.

If the selector is not matched or is not supplied, all previous siblings will be selected; in these cases it selects the same elements as the

`.prevAll()` method does when no filter selector is provided.

**As of jQuery 1.6**, A DOM node or jQuery object, instead of a selector, may be used for the first `.prevUntil()` argument.

The method optionally accepts a selector expression for its second argument. If this argument is supplied, the elements will be filtered by testing whether they match it.

## Example:

*Find the siblings that precede `<dt id="term-2">` up to the preceding `<dt>` and give them a red background color. Also find previous `<dd>` siblings of `<dt id="term-3">` up to `<dt id="term-1">` and give them a green text color.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-
5 </head>
6 <body>
7   <dl>
8     <dt id="term-1">term 1</dt>
9     <dd>definition 1-a</dd>
10    <dd>definition 1-b</dd>
11    <dd>definition 1-c</dd>
12    <dd>definition 1-d</dd>
13
14    <dt id="term-2">term 2</dt>
15    <dd>definition 2-a</dd>
16    <dd>definition 2-b</dd>
17    <dd>definition 2-c</dd>
18
19    <dt id="term-3">term 3</dt>
20    <dd>definition 3-a</dd>
21    <dd>definition 3-b</dd>
22  </dl>
23  <script>
24    $("#term-2").prevUntil("dt")
25      .css("background-color", "red");
26
27    var term1 = document.getElementById('term-1')
```

```
28 $( "#term-3" ).prevUntil( term1, "dd" )
29   .css( "color", "green" );
30 </script>
31
32 </body>
33 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# .promise()

Categories: [Deferred Object](#)

`.promise( [type ] [, target ] )`

Returns: [Promise](#)

**Description:** *Return a Promise object to observe when all actions of a certain type bound to the collection, queued or not, have finished.*

`.promise( [type ] [, target ] )`

version added: [1.6](#)

**type** (default: `fx`)

Type: [String](#)

The type of queue that needs to be observed.

**target**

Type: [PlainObject](#)

Object onto which the promise methods have to be attached

The `.promise()` method returns a dynamically generated Promise that is resolved once all actions of a certain type bound to the collection, queued or not, have ended.

By default, `type` is `"fx"`, which means the returned Promise is resolved when all animations of the selected elements have completed.

Resolve context and sole argument is the collection onto which `.promise()` has been called.

If `target` is provided, `.promise()` will attach the methods onto it and then return this object rather than create a new one. This can be useful to attach the Promise behavior to an object that already exists.

**Note:** *The returned Promise is linked to a Deferred object stored on the `.data()` for an element. Since the `.remove()` method removes the element's data as well as the element*

*itself, it will prevent any of the element's unresolved Promises from resolving. If it is necessary to remove an element from the DOM before its Promise is resolved, use `.detach()` instead and follow with `.removeData()` after resolution.*

## Examples:

**Example:** Using `promise()` on a collection with no active animation returns a resolved Promise:

```
1 | var div = $( "<div />" );
2 |
3 | div.promise().done(function( arg1 ) {
4 |     // will fire right away and alert "true"
5 |     alert( this === div && arg1 === div );
6 | });
```

**Example:** Resolve the returned Promise when all animations have ended (including those initiated in the animation callback or added later on):

```
1 | <!DOCTYPE html>
2 | <html>
3 | <head>
4 |   <style>
5 |     div {
6 |       height: 50px; width: 50px;
7 |       float: left; margin-right: 10px;
8 |       display: none; background-color: #090;
9 |     }
10 | </style>
11 |   <script src="http://code.jquery.com/jquery-
12 | </head>
13 | <body>
14 |
15 | <button>Go</button>
16 | <p>Ready...</p>
17 | <div></div>
18 | <div></div>
```

```

19 <div></div>
20 <div></div>
21
22
23 <script>
24 $("button").on( "click", function() {
25     $("p").append( "Started..." );
26
27     $("div").each(function( i ) {
28         $( this ).fadeIn().fadeOut( 1000 * (i+1) );
29     });
30
31     $("div").promise().done(function() {
32         $("p").append( " Finished! " );
33     });
34 });
35 </script>
36
37 </body>
38 </html>

```

## Demo

**Example:** Resolve the returned Promise using a `$.when()` statement (the `promise()` method makes it possible to do this with jQuery collections):

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5   div {
6     height: 50px; width: 50px;
7     float: left; margin-right: 10px;
8     display: none; background-color: #090;
9   }

```

```
10 </style>
11 <script src="http://code.jquery.com/jquery-latest.js"
12 </head>
13 <body>
14
15 <button>Go</button>
16 <p>Ready...</p>
17 <div></div>
18 <div></div>
19 <div></div>
20 <div></div>
21
22
23 <script>
24 var effect = function() {
25     return $("div").fadeIn(800).delay(1200).fadeOut();
26 };
27
28 $("button").on( "click", function() {
29     $("p").append( " Started... ");
30
31     $.when( effect() ).done(function() {
32         $("p").append( " Finished! ");
33     });
34 });
35
36 </script>
37
38 </body>
39 </html>
```

Demo

**A new version of this book is available!**

---

# .prop()

Categories: [Attributes](#) | [Manipulation](#) > [General Attributes](#)

Get the value of a property for the first element in the set of matched elements or set one or more properties for every matched element.

## Contents:

[.prop\( propertyName \)](#)

.prop( propertyName )

[.prop\( propertyName, value \)](#)

.prop( propertyName, value )

.prop( properties )

.prop( propertyName, function(index, oldPropertyValue) )

`.prop( propertyName )`

Returns: [String](#)

**Description:** *Get the value of a property for the first element in the set of matched elements.*

`.prop( propertyName )`

version added: [1.6](#)

**propertyName**

Type: [String](#)

The name of the property to get.

The `.prop()` method gets the property value for only the *first* element in the matched set. It returns `undefined` for the value of a property that has not been set, or if the matched set has no elements. To get the value for each element individually, use a looping construct such as jQuery's `.each()` or `.map()` method.

The difference between *attributes* and *properties* can be important in specific situations. **Before jQuery 1.6**, the `.attr()` method sometimes took property values into account when retrieving some attributes, which could cause inconsistent behavior. **As of jQuery 1.6**, the `.prop()` method provides a way to explicitly retrieve property values, while `.attr()` retrieves attributes.

For example, `selectedIndex`, `tagName`, `nodeName`, `nodeType`, `ownerDocument`, `defaultChecked`, and `defaultSelected` should be retrieved and set with the `.prop()` method. Prior to jQuery 1.6, these properties were retrievable with the `.attr()` method, but this was not within the scope of `attr`. These do not have corresponding attributes and are only properties.

Concerning boolean attributes, consider a DOM element defined by the HTML markup `<input type="checkbox" checked="checked" />`, and assume it is in a JavaScript variable named `elem`:

`elem.checked`

`true` (Boolean) Will change with checkbox state

`$(elem).prop("checked")`

`true` (Boolean) Will change with checkbox state

<code>elem.getAttribute("checked")</code>	"checked" (String) Initial state of the checkbox; does not change
<code>\$(elem).attr("checked")</code> (1.6)	"checked" (String) Initial state of the checkbox; does not change
<code>\$(elem).attr("checked")</code> (1.6.1+)	"checked" (String) Will change with checkbox state
<code>\$(elem).attr("checked")</code> (pre-1.6)	true (Boolean) Changed with checkbox state

According to the [W3C forms specification](#), the `checked` attribute is a *boolean attribute*, which means the corresponding property is true if the attribute is present at all—even if, for example, the attribute has no value or an empty string value. The preferred cross-browser-compatible way to determine if a checkbox is checked is to check for a "truthy" value on the element's property using one of the following:

```
if ( elem.checked )
if ( $(elem).prop("checked") )
if ( $(elem).is(":checked") )
```

If using jQuery 1.6, the code `if ( $(elem).attr("checked") )` will retrieve the actual content *attribute*, which does not change as the checkbox is checked and unchecked. It is meant only to store the default or initial value of the checked property. To maintain backwards compatibility, the `.attr()` method in jQuery 1.6.1+ will retrieve and update the property for you so no code for boolean attributes is required to be changed to `.prop()`. Nevertheless, the preferred way to retrieve a checked value is with one of the options listed above. To see how this works in the latest jQuery, check/uncheck the checkbox in the example below.

## Additional Notes:

In Internet Explorer prior to version 9, using `.prop()` to set a DOM element property to anything other than a simple primitive value (number, string, or boolean) can cause memory leaks if the property is not removed (using `.removeProp()`) before the DOM element is removed from the document. To safely set values on DOM objects without memory leaks, use `.data()`.



## Example:

*Display the checked property and attribute of a checkbox as it changes.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     p { margin: 20px 0 0 }
6     b { color: blue; }
7   </style>
8   <script src="http://code.jquery.com/jquery-1.10.2.min.js"></script>
9 </head>
10 <body>
11
12 <input id="check1" type="checkbox" checked="" />
13 <label for="check1">Check me</label>
14 <p></p>
15
16 <script>
17 $( "input" ).change( function() {
18   var $input = $( this );
19   $( "p" ).html( ".attr('checked'): <b>" + $input.attr('checked') +
20     + ".prop('checked'): <b>" + $input.prop('checked') +
21     + ".is(':checked'): <b>" + $input.is(':checked') +
22   }).change();
23 </script>
24
25 </body>
26 </html>
```

Demo



`.prop( propertyName, value )`

Returns: [jQuery](#)

**Description:** *Set one or more properties for the set of matched elements.*

**.prop( propertyName, value )**

version added: [1.6](#)

**propertyName**

Type: [String](#)

The name of the property to set.

**value**

Type: [String](#) or [Number](#) or [Boolean](#)

A value to set for the property.

**.prop( properties )**

version added: [1.6](#)

**properties**

Type: [PlainObject](#)

An object of property-value pairs to set.

**.prop( propertyName, function(index, oldPropertyValue) )**

version added: [1.6](#)

**propertyName**

Type: [String](#)

The name of the property to set.

**function(index, oldPropertyValue)**

Type: [Function\(\)](#)

A function returning the value to set. Receives the index position of the element in the set and the old property value as arguments. Within the function, the keyword `this` refers to the current element.

The `.prop()` method is a convenient way to set the value of properties—especially when setting multiple properties, using values returned by a function, or setting values on multiple elements at once. It should be used when setting `selectedIndex`, `tagName`, `nodeName`, `nodeType`, `ownerDocument`, `defaultChecked`, or

`defaultSelected`. Since jQuery 1.6, these properties can no longer be set with the `.attr()` method. They do not have corresponding attributes and are only properties.

Properties generally affect the dynamic state of a DOM element without changing the serialized HTML attribute. Examples include the `value` property of input elements, the `disabled` property of inputs and buttons, or the `checked` property of a checkbox. The `.prop()` method should be used to set disabled and checked instead of the `.attr()` method. The `.val()` method should be used for getting and setting value.

```
1 | $("input").prop("disabled", false);
2 | $("input").prop("checked", true);
3 | $("input").val("someValue");
```

**Important:** the `.removeProp()` method should not be used to set these properties to false. Once a native property is removed, it cannot be added again. See `.removeProp()` for more information.

### Computed property values

By using a function to set properties, you can compute the value based on other properties of the element. For example, to toggle all checkboxes based off their individual values:

```
1 | $("input[type='checkbox']").prop("checked", fu
2 |     return !val;
3 | });
```

**Note:** If nothing is returned in the setter function (ie. `function(index, prop){}`), or if `undefined` is returned, the current value is not changed. This is useful for selectively setting values only when certain criteria are met.

### Additional Notes:

In Internet Explorer prior to version 9, using `.prop()` to set a DOM element property to anything other than a simple primitive value (number, string, or boolean) can cause memory leaks if the property is not removed (using `.removeProp()`) before the DOM element is removed from the document. To safely set values on DOM objects without memory leaks, use `.data()`.

## Example:

*Disable all checkboxes on the page.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5      img { padding:10px; }
6      div { color:red; font-size:24px; }
7    </style>
8    <script src="http://code.jquery.com/jquery-1.10.2.min.js"></script>
9  </head>
10 <body>
11
12   <input type="checkbox" checked="checked" />
13   <input type="checkbox" />
14   <input type="checkbox" />
15   <input type="checkbox" checked="checked" />
16
17   <script>
18     $("input[type='checkbox']").prop({
19       disabled: true
20     });
21   </script>
22
23 </body>
24 </html>
```

Demo

**A new version of this book is available!**

---

# **.pushStack()**

Categories: [Internals](#)

**.pushStack( elements )**

Returns: [jQuery](#)

**Description:** *Add a collection of DOM elements onto the jQuery stack.*

**.pushStack( elements )**

version added: [1.0](#)

**elements**

Type: [Array](#)

An array of elements to push onto the stack and make into a new jQuery object.

**.pushStack( elements, name, arguments )**

version added: [1.3](#)

**elements**

Type: [Array](#)

An array of elements to push onto the stack and make into a new jQuery object.

**name**

Type: [String](#)

The name of a jQuery method that generated the array of elements.

**arguments**

Type: [Array](#)

The arguments that were passed in to the jQuery method (for serialization).

## Example:

*Add some elements onto the jQuery stack, then pop back off again.*

```
1 | jQuery([])  
2 |     .pushStack( document.getElementsByTagName(  
3 |         .remove()  
4 |     .end());
```

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# .queue()

Categories: [Effects](#) > [Custom](#) | [Data](#) | [Utilities](#)

Show or manipulate the queue of functions to be executed on the matched elements.

## Contents:

[.queue\( \[queueName \] \)](#)

[.queue\( \[queueName \] \)](#)

[.queue\( \[queueName \], newQueue \)](#)

[.queue\( \[queueName \], newQueue \)](#)

[.queue\( \[queueName \], callback\( next \) \)](#)

---

`.queue( [queueName ] )`

Returns: [Array](#)

**Description:** *Show the queue of functions to be executed on the matched elements.*

`.queue( [queueName ] )`

version added: [1.2](#)

**queueName**

Type: [String](#)

A string containing the name of the queue. Defaults to `fx`, the standard effects queue.

## Example:

*Show the length of the queue.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>div { margin:3px; width:40px; height
5             position:absolute; left:0px; top:60px
6             background:green; display:none; }
7   div.newcolor { background:blue; }
8   p { color:red; } </style>
9   <script src="http://code.jquery.com/jquery-
10 </head>
11 <body>
12
13   <p>The queue length is: <span></span></p>
14   <div></div>
15 <script>
16 var div = $("div");
17
18 function runIt() {
19   div.show("slow");
20   div.animate({left: '+=200'}, 2000);
21   div.slideToggle(1000);
22   div.slideToggle("fast");
23   div.animate({left: '-=200'}, 1500);
24   div.hide("slow");
25   div.show(1200);
26   div.slideUp("normal", runIt);
27 }
28
29 function showIt() {
30   var n = div.queue("fx");
31   $("span").text( n.length );
```

```
32     setTimeout(showIt, 100);
33 }
34
35 runIt();
36 showIt();
37 </script>
38
39 </body>
40 </html>
```

Demo

`.queue( [queueName ], newQueue )`

Returns: [jQuery](#)

**Description:** *Manipulate the queue of functions to be executed, once for each matched element.*

`.queue( [queueName ], newQueue )`

version added: [1.2](#)

**queueName**

Type: [String](#)

A string containing the name of the queue. Defaults to `fx`, the standard effects queue.

**newQueue**

Type: [Array](#)

An array of functions to replace the current queue contents.

`.queue( [queueName ], callback( next`

version added: [1.2](#)

`)`

**queueName**

Type: [String](#)

A string containing the name of the queue. Defaults to `fx`, the standard effects queue.

**callback( next )**

Type: [Function\(\)](#)

The new function to add to the queue, with a function to call that will dequeue the next item.

Every element can have one to many queues of functions attached to it by jQuery. In most applications, only one queue (called `fx`) is used. Queues allow a sequence of actions to be called on an element asynchronously, without halting program execution. The typical example of this is calling multiple animation methods on an element. For example:

```
1 | $('#foo').slideUp().fadeIn();
```

When this statement is executed, the element begins its sliding animation immediately, but the fading transition is placed on the `fx` queue to be called only once the sliding transition is complete.

The `.queue()` method allows us to directly manipulate this queue of functions. Calling `.queue()` with a callback is particularly useful; it allows us to place a new function at the end of the queue. The callback function is executed once for each element in the jQuery set.

This feature is similar to providing a callback function with an animation method, but does not require the callback to be given at the time the animation is performed.

```
1 | $('#foo').slideUp();
2 | $('#foo').queue(function() {
3 |     alert('Animation complete.');
```

```
4 |     $(this).dequeue();
5 | });
```

This is equivalent to:

```
1 | $('#foo').slideUp(function() {
2 |     alert('Animation complete.');
```

```
3 | });
```

Note that when adding a function with `.queue()`, we should ensure that `.dequeue()` is eventually called so that the next function in line executes.

**As of jQuery 1.4**, the function that's called is passed another function as the first argument. When called, this automatically dequeues the next item and keeps the queue moving. We use it as follows:

```
1 | $("#test").queue(function(next) {
```

```
2 | // Do some stuff...
3 | next();
4 | });
```

## Examples:

**Example:** *Queue a custom function.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5      div { margin:3px; width:40px; height:40px;
6            position:absolute; left:0px; top:30px
7            background:green; display:none; }
8      div.newcolor { background:blue; }
9    </style>
10   <script src="http://code.jquery.com/jquery-
11 </head>
12 <body>
13   Click here...
14   <div></div>
15   <script>$(document.body).click(function () {
16     $("div").show("slow");
17     $("div").animate({left:'+=200'}, 2000);
18     $("div").queue(function () {
19       $(this).addClass("newcolor");
20       $(this).dequeue();
21     });
22     $("div").animate({left:'-=200'}, 500);
23     $("div").queue(function () {
24       $(this).removeClass("newcolor");
25       $(this).dequeue();
26     });
27     $("div").slideUp();
28   });</script>
29
30 </body>
31 </html>
```

## Demo

**Example:** *Set a queue array to delete the queue.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     div { margin:3px; width:40px; height:40px;
6           position:absolute; left:0px; top:30px
7           background:green; display:none; }
8     div.newcolor { background:blue; }
9   </style>
10  <script src="http://code.jquery.com/jquery-
11 </head>
12 <body>
13   <button id="start">Start</button>
14   <button id="stop">Stop</button>
15   <div></div>
16  <script>$("#start").click(function () {
17    $("div").show("slow");
18    $("div").animate({left:'+=200'}, 5000);
19    $("div").queue(function () {
20      $(this).addClass("newcolor");
21      $(this).dequeue();
22    });
23    $("div").animate({left:'-=200'}, 1500);
24    $("div").queue(function () {
25      $(this).removeClass("newcolor");
26      $(this).dequeue();
27    });
28    $("div").slideUp();
29  });
30  $("#stop").click(function () {
31    $("div").queue("fx", []);
```

```
32     $("div").stop();
33     });</script>
34
35 </body>
36 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# :radio Selector

Categories: [Selectors](#) > [Form](#) | [Selectors](#) > [jQuery Extensions](#)

## radio selector

**Description:** *Selects all elements of type radio.*

**jQuery( ":radio" )**

version added: [1.0](#)

`$('.radio')` is equivalent to `$('.[type=radio]')`. As with other pseudo-class selectors (those that begin with a ":") it is recommended to precede it with a tag name or some other selector; otherwise, the universal selector ("\*") is implied. In other words, the bare `$('.radio')` is equivalent to `$('.*:radio')`, so `$('.input:radio')` should be used instead.

To select a set of associated radio buttons, you might use:

```
$('.input[name=gender]:radio')
```

## Additional Notes:

Because `:radio` is a jQuery extension and not part of the CSS specification, queries using `:radio` cannot take advantage of the performance boost provided by the native DOM `querySelectorAll()` method. For better performance in modern browsers, use `[type="radio"]` instead.

## Example:

*Finds all radio inputs.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5      textarea { height:25px; }
6    </style>
7    <script src="http://code.jquery.com/jquery-
8  </head>
9  <body>
10   <form>
11     <input type="button" value="Input Button" />
12     <input type="checkbox" />
13
14     <input type="file" />
15     <input type="hidden" />
16     <input type="image" />
17
18     <input type="password" />
19     <input type="radio" name="asdf" />
20     <input type="radio" name="asdf" />
21
22     <input type="reset" />
23     <input type="submit" />
24     <input type="text" />
25
26     <select><option>Option</option></select>
27     <textarea></textarea>
28     <button>Button</button>
29   </form>
30
31   <div>
```

```
32 </div>
33 <script>
34 var input = $("form input:radio")
35     .wrap('<span></span>')
36     .parent()
37     .css({background:"yellow", border:"3px red solid"});
38
39 $("div").text("For this type jQuery found " + input.length);
40     .css("color", "red");
41 $("form").submit(function () { return false; });
42 </script>
43
44 </body>
45 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# .ready()

Categories: [Events](#) > [Document Loading](#)

`.ready( handler )`

Returns: [jQuery](#)

**Description:** *Specify a function to execute when the DOM is fully loaded.*

`.ready( handler )`

version added: [1.0](#)

**handler**

Type: [Function\(\)](#)

A function to execute after the DOM is ready.

While JavaScript provides the `load` event for executing code when a page is rendered, this event does not get triggered until all assets such as images have been completely received. In most cases, the script can be run as soon as the DOM hierarchy has been fully constructed. The handler passed to `.ready()` is guaranteed to be executed after the DOM is ready, so this is usually the best place to attach all other event handlers and run other jQuery code. When using scripts that rely on the value of CSS style properties, it's important to reference external stylesheets or embed style elements before referencing the scripts.

In cases where code relies on loaded assets (for example, if the dimensions of an image are required), the code should be placed in a handler for the `load` event instead.

*The `.ready()` method is generally incompatible with the `<body onload="">` attribute. If `load` must be used, either do not use `.ready()` or use jQuery's `.load()` method to attach `load` event handlers to the window or to more specific items, like images.*

All three of the following syntaxes are equivalent:

```
$(document).ready(handler)
```

```
$(document).ready(handler) (this is not recommended)
```

```
$(handler)
```

There is also `$(document).on("ready", handler)`, *deprecated as of jQuery 1.8*. This behaves similarly to the ready method but if the ready event has already fired and you try to `.on("ready")` the bound handler will not be executed. Ready handlers bound this way are executed *after* any bound by the other three methods above.

The `.ready()` method can only be called on a jQuery object matching the current document, so the selector can be omitted.

The `.ready()` method is typically used with an anonymous function:

```
1 | $(document).ready(function() {  
2 |     // Handler for .ready() called.  
3 | });
```

Which is equivalent to calling:

```
1 | $(function() {  
2 |     // Handler for .ready() called.  
3 | });
```

If `.ready()` is called after the DOM has been initialized, the new handler passed in will be executed immediately.

## Aliasing the jQuery Namespace

When using another JavaScript library, we may wish to call `$.noConflict()` to avoid namespace difficulties. When this function is called, the `$` shortcut is no longer available, forcing us to write `jquery` each time we would normally write `$`. However, the handler passed to the `.ready()` method can take an argument, which is passed the global `jquery` object. This means we can rename the object within the context of our `.ready()` handler without affecting

other code:

```
1 | jQuery(document).ready(function($) {  
2 |     // Code using $ as usual goes here.  
3 | });
```

## Example:

*Display a message when the DOM is loaded.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <style>p { color:red; }</style>
5      <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
6      <script>
7          $(document).ready(function () {
8              $("p").text("The DOM is now loaded and can be manipulated.");
9          });
10     </script>
11
12 </head>
13 <body>
14     <p>Not loaded yet.</p>
15
16 </body>
17 </html>
```

Demo

POWERED BY HERONOTE

A new version of this book is available!

---

# .remove()

Categories: [Manipulation](#) > [DOM Removal](#)

`.remove( [selector ] )`

Returns: [jQuery](#)

**Description:** *Remove the set of matched elements from the DOM.*

`.remove( [selector ] )`

version added: [1.0](#)

**selector**

Type: [String](#)

A selector expression that filters the set of matched elements to be removed.

Similar to `.empty()`, the `.remove()` method takes elements out of the DOM. Use `.remove()` when you want to remove the element itself, as well as everything inside it. In addition to the elements themselves, all bound events and jQuery data associated with the elements are removed. To remove the elements without removing data and events, use `.detach()` instead.

Consider the following HTML:

```
1 | <div class="container">
2 |   <div class="hello">Hello</div>
3 |   <div class="goodbye">Goodbye</div>
4 | </div>
```

We can target any element for removal:

```
1 | $(' .hello' ).remove( );
```

This will result in a DOM structure with the `<div>` element deleted:

```
1 | <div class="container">
2 |   <div class="goodbye">Goodbye</div>
```

```
3 | </div>
```

If we had any number of nested elements inside `<div class="hello">`, they would be removed, too. Other jQuery constructs such as data or event handlers are erased as well.

We can also include a selector as an optional parameter. For example, we could rewrite the previous DOM removal code as follows:

```
1 | $('div').remove('.hello');
```

This would result in the same DOM structure:

```
1 | <div class="container">  
2 |   <div class="goodbye">Goodbye</div>  
3 | </div>
```

## Examples:

**Example:** *Removes all paragraphs from the DOM*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>p { background:yellow; margin:6px 0;
5   <script src="http://code.jquery.com/jquery-
6 </head>
7 <body>
8   <p>Hello</p>
9   how are
10  <p>you?</p>
11  <button>Call remove() on paragraphs</button>
12 <script>
13  $("button").click(function () {
14    $("p").remove();
15  });
16 </script>
17
18 </body>
19 </html>
```

Demo

**Example:** *Removes all paragraphs that contain "Hello" from the DOM Analogous to doing*

```
$("p").filter(":contains('Hello')").remove();
```

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>p { background:yellow; margin:6px 0;
```

```
5 <script src="http://code.jquery.com/jquery-latest.js"
6 </head>
7 <body>
8 <p class="hello">Hello</p>
9   how are
10  <p>you?</p>
11
12  <button>Call remove(":contains('Hello')") on paragraph
13  <script>
14  $("button").click(function () {
15    $("p").remove(":contains('Hello')");
16  });
17  </script>
18
19  </body>
20  </html>
```

Demo

POWERED BY HERONOTE

A new version of this book is available!

---

# .removeAttr()

Categories: [Attributes](#) | [Manipulation](#) > [General Attributes](#)

`.removeAttr( attributeName )`

Returns: [jQuery](#)

**Description:** *Remove an attribute from each element in the set of matched elements.*

`.removeAttr( attributeName )`

version added: [1.0](#)

**attributeName**

Type: [String](#)

An attribute to remove; as of version 1.7, it can be a space-separated list of attributes.

The `.removeAttr()` method uses the JavaScript `removeAttribute()` function, but it has the advantage of being able to be called directly on a jQuery object and it accounts for different attribute naming across browsers.

**Note:** Removing an inline `onclick` event handler using `.removeAttr()` doesn't achieve the desired effect in Internet Explorer 6, 7, or 8. To avoid potential problems, use `.prop()` instead:

```
1 | $element.prop("onclick", null);  
2 | console.log("onclick property: ", $element[0])
```

## Example:

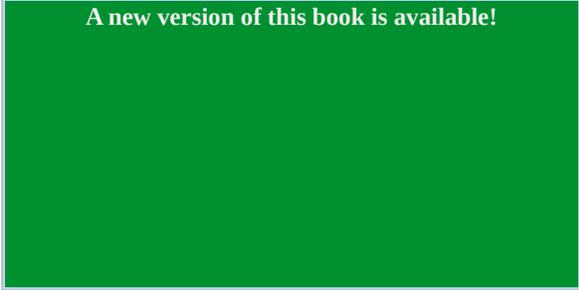
*Clicking the button enables the input next to it.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
5  </head>
6  <body>
7    <button>Enable</button>
8    <input type="text" title="hello there" />
9    <div id="log"></div>
10
11  <script>
12    (function() {
13      var inputTitle = $("input").attr("title");
14      $("button").click(function () {
15        var input = $(this).next();
16
17        if ( input.attr("title") == inputTitle )
18          input.removeAttr("title")
19        } else {
20          input.attr("title", inputTitle);
21        }
22
23        $("#log").html( "input title is now " + input.attr("title")
24      });
25    })();
26  </script>
27
28  </body>
29  </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

# .removeClass()

Categories: [Attributes](#) | [Manipulation](#) > [Class Attribute](#) | [CSS](#)

`.removeClass( [className ] )`

Returns: [jQuery](#)

**Description:** *Remove a single class, multiple classes, or all classes from each element in the set of matched elements.*

`.removeClass( [className ] )`

version added: [1.0](#)

**className**

Type: [String](#)

One or more space-separated classes to be removed from the class attribute of each matched element.

`.removeClass( function(index, class) )`

version added: [1.4](#)

**function(index, class)**

Type: [Function\(\)](#)

A function returning one or more space-separated class names to be removed. Receives the index position of the element in the set and the old class value as arguments.

If a class name is included as a parameter, then only that class will be removed from the set of matched elements. If no class names are specified in the parameter, all classes will be removed.

More than one class may be removed at a time, separated by a space, from the set of matched elements, like so:

```
1 | $( 'p' ).removeClass( 'myClass yourClass' )
```

This method is often used with `.addClass()` to switch elements' classes from one to another, like so:

```
1 | $( 'p' ).removeClass( 'myClass noClass' ).addClass
```

Here, the `myClass` and `noClass` classes are removed from all paragraphs, while `yourClass` is added.

To replace all existing classes with another class, we can use `.attr('class', 'newClass')` instead.

As of jQuery 1.4, the `.removeClass()` method allows us to indicate the class to be removed by passing in a function.

```
1 | $( 'li:last' ).removeClass( function() {  
2 |     return $( this ).prev().attr( 'class' );  
3 | } );
```

This example removes the class name of the penultimate `<li>` from the last `<li>`.

## Examples:

**Example:** Remove the class 'blue' from the matched elements.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5
6   p { margin: 4px; font-size:16px; font-weight: bold; }
7   .blue { color:blue; }
8   .under { text-decoration:underline; }
9   .highlight { background:yellow; }
10 </style>
11 <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
12 </head>
13 <body>
14   <p class="blue under">Hello</p>
15   <p class="blue under highlight">and</p>
16   <p class="blue under">then</p>
17
18   <p class="blue under">Goodbye</p>
19 <script>$("p:even").removeClass("blue");</script>
20
21 </body>
22 </html>
```

Demo

**Example:** Remove the class 'blue' and 'under' from the matched elements.

```
1 <!DOCTYPE html>
```

```

2 <html>
3 <head>
4   <style>
5     p { margin: 4px; font-size:16px; font-weight:
6     .blue { color:blue; }
7     .under { text-decoration:underline; }
8     .highlight { background:yellow; }
9   </style>
10  <script src="http://code.jquery.com/jquery-latest.js"
11 </head>
12 <body>
13   <p class="blue under">Hello</p>
14
15   <p class="blue under highlight">and</p>
16   <p class="blue under">then</p>
17   <p class="blue under">Goodbye</p>
18 <script>$("p:odd").removeClass("blue under");</
19
20 </body>
21 </html>

```

Demo

**Example:** *Remove all the classes from the matched elements.*

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5
6     p { margin: 4px; font-size:16px; font-weight:
7     .blue { color:blue; }
8     .under { text-decoration:underline; }
9     .highlight { background:yellow; }
10  </style>

```

```
11 <script src="http://code.jquery.com/jquery-latest.js"
12 </head>
13 <body>
14 <p class="blue under">Hello</p>
15 <p class="blue under highlight">and</p>
16 <p class="blue under">then</p>
17
18 <p class="blue under">Goodbye</p>
19 <script>$("p:eq(1)").removeClass();</script>
20
21 </body>
22 </html>
```

Demo

POWERED BYHERONOTE

A new version of this book is available!

---

# `.removeData()`

Categories: [Data](#) | [Miscellaneous](#) > [Data Storage](#)

`.removeData( [name ] )`

Returns: [jQuery](#)

**Description:** *Remove a previously-stored piece of data.*

`.removeData( [name ] )`

version added: [1.2.3](#)

**name**

Type: [String](#)

A string naming the piece of data to delete.

`.removeData( [list ] )`

version added: [1.7](#)

**list**

Type: [Array](#) or [String](#)

An array or space-separated string naming the pieces of data to delete.

The `.removeData()` method allows us to remove values that were previously set using `.data()`. When called with the name of a key, `.removeData()` deletes that particular value; when called with no arguments, all values are removed. Removing data from jQuery's internal `.data()` cache does not effect any HTML5 `data-` attributes in a document; use `.removeAttr()` to remove those.

When using `.removeData("name")`, jQuery will attempt to locate a `data-` attribute on the element if no property by that name is in the internal data cache. To avoid a re-query of the `data-` attribute, set the name to a value of either `null` or `undefined` (e.g. `.data("name", undefined)`) rather than using `.removeData()`.

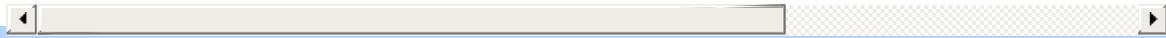
**As of jQuery 1.7**, when called with an array of keys or a string of space-separated keys, `.removeData()` deletes the value of each key in that array or string.

**As of jQuery 1.4.3**, calling `.removeData()` will cause the value of the property being removed to revert to the value of the data attribute of the same name in the DOM, rather than being set to `undefined`.

## Example:

*Set a data store for 2 names then remove one of them.*

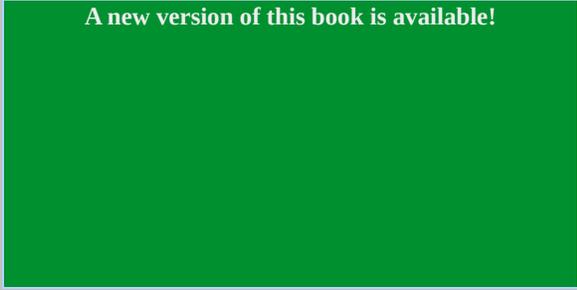
```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     div { margin:2px; color:blue; }
6     span { color:red; }
7   </style>
8   <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
9 </head>
10 <body>
11   <div>value1 before creation: <span></span></div>
12   <div>value1 after creation: <span></span></div>
13   <div>value1 after removal: <span></span></div>
14
15   <div>value2 after removal: <span></span></div>
16 <script>
17
18   $("span:eq(0)").text(" " + $("div").data("test1"));
19   $("div").data("test1", "VALUE-1");
20   $("div").data("test2", "VALUE-2");
21   $("span:eq(1)").text(" " + $("div").data("test2"));
22   $("div").removeData("test1");
23   $("span:eq(2)").text(" " + $("div").data("test2"));
24   $("span:eq(3)").text(" " + $("div").data("test2"));
25
26 </script>
27
28 </body>
29 </html>
```



Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

# .removeProp()

Categories: [Attributes](#) | [Manipulation](#) > [General Attributes](#)

`.removeProp( propertyName )`

Returns: [jQuery](#)

**Description:** *Remove a property for the set of matched elements.*

`.removeProp( propertyName )`

version added: [1.6](#)

**propertyName**

Type: [String](#)

The name of the property to remove.

The `.removeProp()` method removes properties set by the `.prop()` method.

With some built-in properties of a DOM element or `window` object, browsers may generate an error if an attempt is made to remove the property. jQuery first assigns the value `undefined` to the property and ignores any error the browser generates. In general, it is only necessary to remove custom properties that have been set on an object, and not built-in (native) properties.

**Note:** Do not use this method to remove native properties such as `checked`, `disabled`, or `selected`. This will remove the property completely and, once removed, cannot be added again to element. Use `.prop()` to set these properties to `false` instead.

## Additional Notes:

In Internet Explorer prior to version 9, using `.prop()` to set a DOM element property to anything other than a simple primitive value (number, string, or boolean) can cause memory leaks if the property is not removed (using `.removeProp()`) before the DOM element is removed from the document. To safely set values on DOM objects without memory leaks, use `.data()`.

## Example:

*Set a numeric property on a paragraph and then remove it.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     img { padding:10px; }
6     div { color:red; font-size:24px; }
7   </style>
8   <script src="http://code.jquery.com/jquery-1.10.2.min.js"></script>
9 </head>
10 <body>
11
12   <p></p>
13
14   <script>
15   var $para = $("p");
16   $para.prop("luggageCode", 1234);
17   $para.append("The secret luggage code is: ", $para.prop("luggageCode"));
18   $para.removeProp("luggageCode");
19   $para.append("Now the secret luggage code is: ");
20
21   </script>
22
23 </body>
24 </html>
```

Demo

A new version of this book is available!

---

# .replaceAll()

Categories: [Manipulation](#) > [DOM Replacement](#)

`.replaceAll( target )`

Returns: [jQuery](#)

**Description:** *Replace each target element with the set of matched elements.*

`.replaceAll( target )`

version added: [1.2](#)

**target**

Type: [Selector](#)

A selector expression indicating which element(s) to replace.

The `.replaceAll()` method is corollary to `.replaceWith()`, but with the source and target reversed. Consider this DOM structure:

```
1 | <div class="container">
2 |   <div class="inner first">Hello</div>
3 |   <div class="inner second">And</div>
4 |   <div class="inner third">Goodbye</div>
5 | </div>
```

We can create an element, then replace other elements with it:

```
1 | $( '<h2>New heading</h2>' ).replaceAll( '.inner'
```

This causes all of them to be replaced:

```
1 | <div class="container">
2 |   <h2>New heading</h2>
3 |   <h2>New heading</h2>
4 |   <h2>New heading</h2>
5 | </div>
```

Or, we could select an element to use as the replacement:

```
1 | $(' .first' ).replaceAll( ' .third' );
```

This results in the DOM structure:

```
1 | <div class="container">  
2 |   <div class="inner second">And</div>  
3 |   <div class="inner first">Hello</div>  
4 | </div>
```

From this example, we can see that the selected element replaces the target by being moved from its old location, not by being cloned.

## Example:

*Replace all the paragraphs with bold words.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <script src="http://code.jquery.com/jquery-
5  </head>
6  <body>
7    <p>Hello</p>
8    <p>cruel</p>
9    <p>World</p>
10 <script>$("<b>Paragraph. </b>").replaceAll("p
11
12 </body>
13 </html>
```

Demo

POWERED BY **HERONOTE**

A new version of this book is available!

---

# .replaceWith()

Categories: [Manipulation](#) > [DOM Replacement](#)

`.replaceWith( newContent )`

Returns: [jQuery](#)

**Description:** *Replace each element in the set of matched elements with the provided new content and return the set of elements that was removed.*

`.replaceWith( newContent )`

version added: [1.2](#)

**newContent**

Type: [htmlString](#) or [Element](#) or [jQuery](#)

The content to insert. May be an HTML string, DOM element, or jQuery object.

`.replaceWith( function )`

version added: [1.4](#)

**function**

Type: [Function\(\)](#)

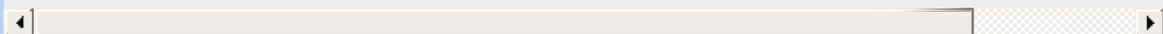
A function that returns content with which to replace the set of matched elements.

The `.replaceWith()` method removes content from the DOM and inserts new content in its place with a single call. Consider this DOM structure:

```
1 | <div class="container">
2 |   <div class="inner first">Hello</div>
3 |   <div class="inner second">And</div>
4 |   <div class="inner third">Goodbye</div>
5 | </div>
```

The second inner `<div>` could be replaced with the specified HTML:

```
1 | $('div.second').replaceWith('<h2>New heading<
```



This results in the structure:

```
1 | <div class="container">
2 |   <div class="inner first">Hello</div>
3 |   <h2>New heading</h2>
4 |   <div class="inner third">Goodbye</div>
5 | </div>
```

All inner `<div>` elements could be targeted at once:

```
1 | $('div.inner').replaceWith('<h2>New heading</h2>')
```

This causes all of them to be replaced:

```
1 | <div class="container">
2 |   <h2>New heading</h2>
3 |   <h2>New heading</h2>
4 |   <h2>New heading</h2>
5 | </div>
```

An element could also be selected as the replacement:

```
1 | $('div.third').replaceWith($('.first'));
```

This results in the DOM structure:

```
1 | <div class="container">
2 |   <div class="inner second">And</div>
3 |   <div class="inner first">Hello</div>
4 | </div>
```

This example demonstrates that the selected element replaces the target by being moved from its old location, not by being cloned.

The `.replaceWith()` method, like most jQuery methods, returns the jQuery object so that other methods can be chained onto it. However, it must be noted that the *original* jQuery object is returned. This object refers to the element that has been removed from the DOM, not the new element that has replaced it.

As of jQuery 1.4, `.replaceWith()` can also work on disconnected DOM nodes. For example, with the following code, `.replaceWith()` returns a jQuery set containing only a paragraph.:

```
1 | $("<div/>").replaceWith("<p></p>");
```

The `.replaceWith()` method can also take a function as its argument:

```
1 | $('div.container').replaceWith(function() {  
2 |   return $(this).contents();  
3 | });
```

This results in `<div class="container">` being replaced by its three child `<div>`s. The return value of the function may be an HTML string, DOM element, or jQuery object.

## Examples:

**Example:** *On click, replace the button with a div containing the same word.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     button { display:block; margin:3px; color:red; }
6     div { color:red; border:2px solid blue; width:100px;
7         margin:3px; text-align:center; }
8   </style>
9   <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
10 </head>
11 <body>
12
13 <button>First</button>
14 <button>Second</button>
15 <button>Third</button>
16
17 <script>
18 $("button").click(function () {
19   $(this).replaceWith( "<div>" + $(this).text() );
20 });
21 </script>
22
23 </body>
24 </html>
```

Demo

**Example:** *Replace all paragraphs with bold words.*

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4    <script src="http://code.jquery.com/jquery-latest.js"
5  </head>
6  <body>
7
8    <p>Hello</p>
9    <p>cruel</p>
10   <p>World</p>
11
12   <script>
13     $("p").replaceWith( "<b>Paragraph. </b>" );
14   </script>
15
16 </body>
17 </html>

```

## Demo

**Example:** *On click, replace each paragraph with a div that is already in the DOM and selected with the `$(...)` function. Notice it doesn't clone the object but rather moves it to replace the paragraph.*

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5      div { border:2px solid blue; color:red; margin:0; padding:5px; }
6      p { border:2px solid red; color:blue; margin:0; padding:5px; }
7    </style>
8    <script src="http://code.jquery.com/jquery-latest.js"
9  </head>
10 <body>
11

```

```

12 <p>Hello</p>
13 <p>cruel</p>
14 <p>World</p>
15
16 <div>Replaced!</div>
17
18 <script>
19 $("p").click(function () {
20   $(this).replaceWith( $("div") );
21 });
22 </script>
23
24 </body>
25 </html>

```

## Demo

**Example:** *On button click, replace the containing div with its child divs and append the class name of the selected element to the paragraph.*

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     .container { background-color: #991; }
6     .inner { color: #911; }
7   </style>
8   <script src="http://code.jquery.com/jquery-
9 </head>
10 <body>
11
12 <p>
13   <button>Replace!</button>
14 </p>
15 <div class="container">

```

```
16 <div class="inner">Scooby</div>
17 <div class="inner">Dooby</div>
18 <div class="inner">Doo</div>
19 </div>
20
21 <script>
22 $('button').on("click", function() {
23   var $container = $("div.container").replaceWith(
24     return $(this).contents();
25   });
26
27   $("p").append( $container.attr("class") );
28 });
29 </script>
30
31 </body>
32 </html>
```

Demo

POWERED BY HERONOTE

A new version of this book is available!

---

# :reset Selector

Categories: [Selectors](#) > [Form](#) | [Selectors](#) > [jQuery Extensions](#)

## reset selector

**Description:** *Selects all elements of type reset.*

**jQuery( ":reset" )**

version added: [1.0](#)

`:reset` is equivalent to `[type="reset"]`

### Additional Notes:

Because `:reset` is a jQuery extension and not part of the CSS specification, queries using `:reset` cannot take advantage of the performance boost provided by the native DOM `querySelectorAll()` method. For better performance in modern browsers, use `[type="reset"]` instead.

## Example:

*Finds all reset inputs.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5      textarea { height:45px; }
6    </style>
7    <script src="http://code.jquery.com/jquery-
8  </head>
9  <body>
10   <form>
11     <input type="button" value="Input Button"
12     <input type="checkbox" />
13
14     <input type="file" />
15     <input type="hidden" />
16     <input type="image" />
17
18     <input type="password" />
19     <input type="radio" />
20     <input type="reset" />
21
22     <input type="submit" />
23     <input type="text" />
24     <select><option>Option</option></select>
25
26     <textarea></textarea>
27     <button>Button</button>
28   </form>
29   <div>
30   </div>
31 </script>
```

```
32 var input = $("input:reset").css({background:"yellow"  
33 $("div").text("For this type jQuery found " + input.length  
34 .css("color", "red");  
35 $("form").submit(function () { return false; });  
36 </script>  
37  
38 </body>  
39 </html>
```

Demo

POWERED BYHERONOTE

A new version of this book is available!

---

# .resize()

Categories: [Events](#) > [Browser Events](#)

`.resize( handler(eventObject) )`

Returns: [jQuery](#)

**Description:** *Bind an event handler to the "resize" JavaScript event, or trigger that event on an element.*

**.resize( handler(eventObject) )**

version added: [1.0](#)

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute each time the event is triggered.

**.resize( [eventData ],  
handler(eventObject) )**

version added: [1.4.3](#)

**eventData**

Type: [PlainObject](#)

An object containing data that will be passed to the event handler.

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute each time the event is triggered.

**.resize()**

version added: [1.0](#)

This method does not accept any arguments.

This method is a shortcut for `.on('resize', handler)` in the first and second variations, and `.trigger('resize')` in the third.

The `resize` event is sent to the `window` element when the size of the browser window changes:

```
1 | $(window).resize(function() {  
2 |     $('#log').append('<div>Handler for .resize(  
3 | });
```

Now whenever the browser window's size is changed, the message is appended to `<div id="log">` one or more times, depending on the browser.

Code in a `resize` handler should never rely on the number of times the handler is called. Depending on implementation, `resize` events can be sent continuously as the resizing is in progress (the typical behavior in Internet Explorer and WebKit-based browsers such as Safari and Chrome), or only once at the end of the resize operation (the typical behavior in some other browsers such as Opera).

---

## Example:

To see the window width while (or after) it is resized, try:

```
1 | $(window).resize(function() {  
2 |     $('body').prepend('<div>' + $(window).width(  
3 | });
```

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# :root Selector

Categories: [Selectors](#) > [Basic Filter](#)

## root selector

**Description:** *Selects the element that is the root of the document.*

### jQuery( ":root" )

version added: [1.9](#)

**index:** The index of each child to match, starting with `1`, the string `even` or `odd`, or an equation ( eg. `:nth-last-child(even)`, `:nth-last-child(4n)` )

In HTML, the root of the document, and thus the element that `$(":root")` selects, is always the `<html>` element.

## Example:

*Display the tag name of the root element.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5    span.fot { color: red; font-size: 120%; font-
6    </style>
7    <script src="http://code.jquery.com/jquery-
8  </head>
9  <body>
10   <div id="log">The root of this document is:
11   <script>
12   $( "<b></b>" ).html( $( ":root" )[0].nodeName
13   </script>
14
15  </body>
16  </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# .scroll()

Categories: [Events](#) > [Browser Events](#)

`.scroll( handler(eventObject) )`

Returns: [jQuery](#)

**Description:** *Bind an event handler to the "scroll" JavaScript event, or trigger that event on an element.*

`.scroll( handler(eventObject) )`

version added: [1.0](#)

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute each time the event is triggered.

`.scroll( [eventData ],  
handler(eventObject) )`

version added: [1.4.3](#)

**eventData**

Type: [PlainObject](#)

An object containing data that will be passed to the event handler.

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute each time the event is triggered.

`.scroll()`

version added: [1.0](#)

This method does not accept any arguments.

This method is a shortcut for `.on('scroll', handler)` in the first and second variations, and `.trigger('scroll')` in the third.

The `scroll` event is sent to an element when the user scrolls to a different place in the element. It applies to `window` objects, but also to scrollable frames and elements with the `overflow` CSS property set to `scroll` (or `auto` when the element's explicit height or width is less than the height or width of its contents).

For example, consider the HTML:

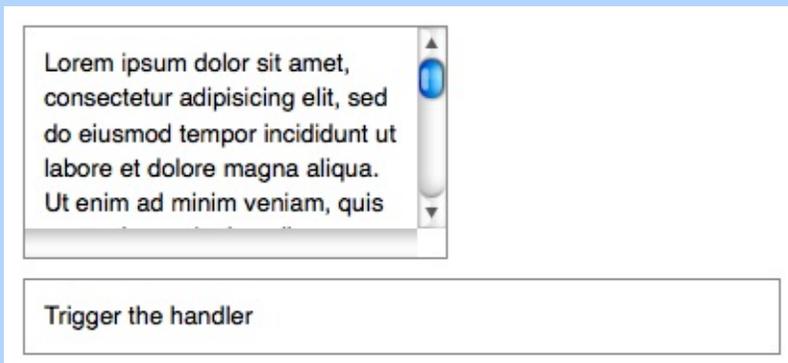
```
1 | <div id="target" style="overflow: scroll; width: 100px; height: 100px; border: 1px solid black;">
```

```

2 | Lorem ipsum dolor sit amet, consectetur adipisicing elit,
3 | sed do eiusmod tempor incididunt ut labore et dolore
4 | aliqua. Ut enim ad minim veniam, quis nostrud exerciti
5 | ullamco laboris nisi ut aliquip ex ea commodo consequ
6 | Duis aute irure dolor in reprehenderit in voluptate v
7 | esse cillum dolore eu fugiat nulla pariatur. Excepteu
8 | sint occaecat cupidatat non proident, sunt in culpa o
9 | officia deserunt mollit anim id est laborum.
10 | </div>
11 | <div id="other">
12 |   Trigger the handler
13 | </div>
14 | <div id="log"></div>

```

The style definition is present to make the target element small enough to be scrollable:



The `scroll` event handler can be bound to this element:

```

1 | $('#target').scroll(function() {
2 |   $('#log').append('<div>Handler for .scroll(
3 | });

```

Now when the user scrolls the text up or down, one or more messages are appended to `<div id="log"></div>`:

Handler for `.scroll()` called.

To trigger the event manually, apply `.scroll()` without an argument:

```
1 | $('#other').click(function() {  
2 |     $('#target').scroll();  
3 | });
```

After this code executes, clicks on Trigger the handler will also append the message.

A `scroll` event is sent whenever the element's scroll position changes, regardless of the cause. A mouse click or drag on the scroll bar, dragging inside the element, pressing the arrow keys, or using the mouse's scroll wheel could cause this event.

## Example:

*To do something when your page is scrolled:*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5    div { color:blue; }
6    p { color:green; }
7    span { color:red; display:none; }
8    </style>
9    <script src="http://code.jquery.com/jquery-1.10.2.js"></script>
10 </head>
11 <body>
12   <div>Try scrolling the iframe.</div>
13   <p>Paragraph - <span>Scroll happened!</span></p>
14   <script>
15     $("p").clone().appendTo(document.body);
16     $("p").clone().appendTo(document.body);
17     $("p").clone().appendTo(document.body);
18     $(window).scroll(function () {
19       $("span").css("display", "inline").fadeOut(1000);
20     });
21   </script>
22
23 </body>
24 </html>
```

Demo

**A new version of this book is available!**

---

# .scrollLeft()

Categories: [CSS](#) | [Offset](#) | [Manipulation](#) > [Style Properties](#)

Get the current horizontal position of the scroll bar for the first element in the set of matched elements or set the horizontal position of the scroll bar for every matched element.

## Contents:

[.scrollLeft\(\)](#)

[.scrollLeft\(\)](#)

[.scrollLeft\( value \)](#)

[.scrollLeft\( value \)](#)

`.scrollLeft()`

Returns: `Integer`

**Description:** *Get the current horizontal position of the scroll bar for the first element in the set of matched elements.*

`.scrollLeft()`

version added: `1.2.6`

This method does not accept any arguments.

The horizontal scroll position is the same as the number of pixels that are hidden from view to the left of the scrollable area. If the scroll bar is at the very left, or if the element is not scrollable, this number will be `0`.

**Note:** `.scrollLeft()`, when called directly or animated as a property using `.animate()`, will not work if the element it is being applied to is hidden.

## Example:

*Get the scrollLeft of a paragraph.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5      p { margin:10px;padding:5px;border:2px so.
6    </style>
7    <script src="http://code.jquery.com/jquery-
8  </head>
9  <body>
10   <p>Hello</p><p></p>
11   <script>var p = $("p:first");
12   $("p:last").text( "scrollLeft:" + p.scrollLef
13 </script>
14
15 </body>
16 </html>
```

Demo

`.scrollLeft( value )`

Returns: [jQuery](#)

**Description:** *Set the current horizontal position of the scroll bar for each of the set of matched elements.*

`.scrollLeft( value )`

version added: [1.2.6](#)

**value**

Type: [Number](#)

An integer indicating the new position to set the scroll bar to.

The horizontal scroll position is the same as the number of pixels that are hidden from view above the scrollable area. Setting the `scrollLeft` positions the horizontal scroll of each matched element.

## Example:

*Set the scrollLeft of a div.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5    div.demo {
6      background:#CCCCCC none repeat scroll 0 0;
7      border:3px solid #666666;
8      margin:5px;
9      padding:5px;
10   position:relative;
11   width:200px;
12   height:100px;
13   overflow:auto;
14   }
15   p { margin:10px;padding:5px;border:2px solid
16   </style>
17   <script src="http://code.jquery.com/jquery-
18 </head>
19 <body>
20   <div class="demo"><h1>lalala</h1><p>Hello</
21   <script>$("div.demo").scrollLeft(300);
22   </script>
23
24 </body>
25 </html>
```

Demo

**A new version of this book is available!**

---

# .scrollTop()

Categories: [CSS](#) | [Offset](#) | [Manipulation](#) > [Style Properties](#)

## Contents:

[.scrollTop\(\)](#)

[.scrollTop\(\)](#)

[.scrollTop\( value \)](#)

[.scrollTop\( value \)](#)

`.scrollTop()`

Returns: [Integer](#)

**Description:** *Get the current vertical position of the scroll bar for the first element in the set of matched elements or set the vertical position of the scroll bar for every matched element.*

`.scrollTop()`

version added: [1.2.6](#)

This method does not accept any arguments.

The vertical scroll position is the same as the number of pixels that are hidden from view above the scrollable area. If the scroll bar is at the very top, or if the element is not scrollable, this number will be `0`.

## Example:

*Get the scrollTop of a paragraph.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5    p { margin:10px;padding:5px;border:2px solid
6    </style>
7    <script src="http://code.jquery.com/jquery-
8  </head>
9  <body>
10   <p>Hello</p><p></p>
11   <script>var p = $("p:first");
12   $("p:last").text( "scrollTop:" + p.scrollTop(
13   </script>
14
15  </body>
16  </html>
```

Demo

`.scrollTop( value )`

Returns: [jQuery](#)

**Description:** *Set the current vertical position of the scroll bar for each of the set of matched elements.*

**.scrollTop( value )**

version added: [1.2.6](#)

**value**

Type: [Number](#)

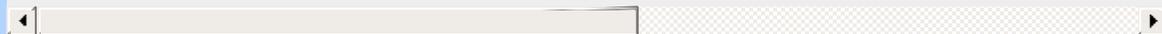
An integer indicating the new position to set the scroll bar to.

The vertical scroll position is the same as the number of pixels that are hidden from view above the scrollable area. Setting the `scrollTop` positions the vertical scroll of each matched element.

## Example:

*Set the scrollTop of a div.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     div.demo {
6       background:#CCCCCC none repeat scroll 0 0;
7       border:3px solid #666666;
8       margin:5px;
9       padding:5px;
10      position:relative;
11      width:200px;
12      height:100px;
13      overflow:auto;
14    }
15    p { margin:10px;padding:5px;border:2px solid
16      </style>
17      <script src="http://code.jquery.com/jquery-
18 </head>
19 <body>
20   <div class="demo"><h1>lalala</h1><p>Hello</
21   <script>$("div.demo").scrollTop(300);
22 </script>
23
24 </body>
25 </html>
```



Demo

**A new version of this book is available!**

---

# .select()

Categories: [Events](#) > [Form Events](#) | [Forms](#)

`.select( handler(eventObject) )`

Returns: [jQuery](#)

**Description:** *Bind an event handler to the "select" JavaScript event, or trigger that event on an element.*

`.select( handler(eventObject) )`

version added: [1.0](#)

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute each time the event is triggered.

`.select( [eventData ],  
handler(eventObject) )`

version added: [1.4.3](#)

**eventData**

Type: [PlainObject](#)

An object containing data that will be passed to the event handler.

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute each time the event is triggered.

`.select()`

version added: [1.0](#)

This method does not accept any arguments.

This method is a shortcut for `.on('select', handler)` in the first two variations, and `.trigger('select')` in the third.

The `select` event is sent to an element when the user makes a text selection inside it. This event is limited to `<input type="text">` fields and `<textarea>` boxes.

For example, consider the HTML:

```
1 | <form>  
2 |   <input id="target" type="text" value="Hello"  
3 | </form>
```

```
4 | <div id="other">
5 |   Trigger the handler
6 | </div>
```

The event handler can be bound to the text input:

```
1 | $('#target').select(function() {
2 |   alert('Handler for .select() called. ');
3 | });
```

Now when any portion of the text is selected, the alert is displayed. Merely setting the location of the insertion point will not trigger the event. To trigger the event manually, apply `.select()` without an argument:

```
1 | $('#other').click(function() {
2 |   $('#target').select();
3 | });
```

After this code executes, clicks on the Trigger button will also alert the message:

Handler for `.select()` called.

In addition, the default `select` action on the field will be fired, so the entire text field will be selected.

*The method for retrieving the current selected text differs from one browser to another. A number of jQuery plug-ins offer cross-platform solutions.*



## Examples:

**Example:** To do something when text in input boxes is selected:

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     p { color:blue; }
6     div { color:red; }
7   </style>
8   <script src="http://code.jquery.com/jquery-1.10.2.min.js"></script>
9 </head>
10 <body>
11
12   <p>
13     Click and drag the mouse to select text in the input boxes below.
14   </p>
15   <input type="text" value="Some text" />
16   <input type="text" value="to test on" />
17
18   <div></div>
19 <script>
20   $( ":input" ).select( function () {
21     $( "div" ).text( "Something was selected" ).show();
22   });
23 </script>
24
25 </body>
26 </html>
```

Demo

**Example** To trigger the select event on all input elements, try:

```
1 | $("input").select();
```

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# :selected Selector

Categories: [Selectors](#) > [Form](#) | [Selectors](#) > [jQuery Extensions](#)

## selected selector

**Description:** *Selects all elements that are selected.*

**jQuery( ":selected" )**

version added: [1.0](#)

The `:selected` selector works for `<option>` elements. It does not work for checkboxes or radio inputs; use `:checked` for them.

### Additional Notes:

Because `:selected` is a jQuery extension and not part of the CSS specification, queries using `:selected` cannot take advantage of the performance boost provided by the native DOM `querySelectorAll()` method. To achieve the best performance when using `:selected` to select elements, first select the elements using a pure CSS selector, then use `.filter(":selected")`.

## Example:

*Attaches a change event to the select that gets the text for each selected option and writes them in the div. It then triggers the event for the initial text draw.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     div { color:red; }
6   </style>
7   <script src="http://code.jquery.com/jquery-
8 </head>
9 <body>
10  <select name="garden" multiple="multiple">
11
12    <option>Flowers</option>
13    <option selected="selected">Shrubs</option>
14    <option>Trees</option>
15    <option selected="selected">Bushes</option>
16
17    <option>Grass</option>
18    <option>Dirt</option>
19  </select>
20  <div></div>
21 <script>
22  $("select").change(function () {
23    var str = "";
24    $("select option:selected").each(function (
25      str += $(this).text() + " ";
26    });
27    $("div").text(str);
28  })
29  .trigger('change');
```

```
30 </script>
31
32 </body>
33 </html>
```

Demo

POWERED BY **HERONOTE**

A new version of this book is available!

---

# .selector

Categories: [Internals](#) | [Properties](#) > [Properties of the Global jQuery Object](#)

`selector`

Returns: [String](#)  
version removed: [1.9](#)

**Description:** *A selector representing selector originally passed to `jQuery()`.*

`selector`

version added: [1.3](#)

Should be used in conjunction with `context` to determine the exact query used.

The `.live()` method for binding event handlers uses this property to determine how to perform its searches. Plug-ins which perform similar tasks may also find the property useful. This property contains a string representing the matched set of elements, but if DOM traversal methods have been called on the object, the string may not be a valid jQuery selector expression. For this reason, the value of `.selector` is generally most useful immediately following the original creation of the object. Consequently, the `.live()` method should only be used in this scenario.

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# .serialize()

Categories: [Forms](#) | [Ajax](#) > [Helper Functions](#)

`.serialize()`

Returns: [String](#)

**Description:** *Encode a set of form elements as a string for submission.*

`.serialize()`

version added: [1.0](#)

This method does not accept any arguments.

The `.serialize()` method creates a text string in standard URL-encoded notation. It operates on a jQuery object representing a set of form elements. The form elements can be of several types:

```
1 <form>
2   <div><input type="text" name="a" value="1"
3   <div><input type="text" name="b" value="2"
4   <div><input type="hidden" name="c" value="3"
5   <div>
6     <textarea name="d" rows="8" cols="40">4</
7   </div>
8   <div><select name="e">
9     <option value="5" selected="selected">5</
10    <option value="6">6</option>
11    <option value="7">7</option>
12  </select></div>
13  <div>
14    <input type="checkbox" name="f" value="8"
15  </div>
16  <div>
17    <input type="submit" name="g" value="Subm
18  </div>
19 </form>
```

The `.serialize()` method can act on a jQuery object that has

selected individual form elements, such as `<input>`, `<textarea>`, and `<select>`. However, it is typically easier to select the `<form>` tag itself for serialization:

```
1 | $('form').submit(function() {  
2 |     alert($(this).serialize());  
3 |     return false;  
4 | });
```

This produces a standard-looking query string:

```
1 | a=1&b=2&c=3&d=4&e=5
```

**Warning:** selecting both the form and its children will cause duplicates in the serialized string.

Note: Only "[successful controls](#)" are serialized to the string. No submit button value is serialized since the form was not submitted using a button. For a form element's value to be included in the serialized string, the element must have a `name` attribute. Values from checkboxes and radio buttons (`input`s of type "radio" or "checkbox") are included only if they are checked. Data from file select elements is not serialized.

## Example:

*Serialize a form to a query string, that could be sent to a server in an Ajax request.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5    body, select { font-size:12px; }
6    form { margin:5px; }
7    p { color:red; margin:5px; font-size:14px; }
8    b { color:blue; }
9    </style>
10   <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
11 </head>
12 <body>
13
14
15 <form>
16   <select name="single">
17     <option>Single</option>
18     <option>Single2</option>
19   </select>
20
21   <br />
22   <select name="multiple" multiple="multiple">
23     <option selected="selected">Multiple</option>
24     <option>Multiple2</option>
25
26     <option selected="selected">Multiple3</option>
27   </select>
28   <br />
29   <input type="checkbox" name="check" value="checked">
30
```

```
31 <label for="ch1">check1</label>
32
33 <input type="checkbox" name="check" value="check2"
34
35 <label for="ch2">check2</label>
36 <br />
37 <input type="radio" name="radio" value="radio1"
38
39 <label for="r1">radio1</label>
40 <input type="radio" name="radio" value="radio2"
41
42 <label for="r2">radio2</label>
43 </form>
44 <p><tt id="results"></tt></p>
45 <script>
46     function showValues() {
47         var str = $("form").serialize();
48         $("#results").text(str);
49     }
50     $(".checkbox, .radio").click(showValues);
51     $(".select").change(showValues);
52     showValues();
53 </script>
54
55 </body>
56 </html>
```

Demo

**A new version of this book is available!**

---

# .serializeArray()

Categories: [Forms](#) | [Ajax](#) > [Helper Functions](#)

`.serializeArray()`

Returns: `Array`

**Description:** *Encode a set of form elements as an array of names and values.*

`.serializeArray()`

version added: [1.2](#)

This method does not accept any arguments.

The `.serializeArray()` method creates a JavaScript array of objects, ready to be encoded as a JSON string. It operates on a jQuery object representing a set of form elements. The form elements can be of several types:

```
1 <form>
2   <div><input type="text" name="a" value="1"
3   <div><input type="text" name="b" value="2"
4   <div><input type="hidden" name="c" value=":"
5   <div>
6     <textarea name="d" rows="8" cols="40">4<
7   </div>
8   <div><select name="e">
9     <option value="5" selected="selected">5<
10    <option value="6">6</option>
11    <option value="7">7</option>
12  </select></div>
13  <div>
14    <input type="checkbox" name="f" value="8"
15  </div>
16  <div>
17    <input type="submit" name="g" value="Subr
18  </div>
19 </form>
```

The `.serializeArray()` method uses the standard W3C rules for [successful controls](#) to determine which elements it should include; in particular the element cannot be disabled and must contain a `name` attribute. No submit button value is serialized since the form was not submitted using a button. Data from file select elements is not serialized.

This method can act on a jQuery object that has selected individual form elements, such as `<input>`, `<textarea>`, and `<select>`. However, it is typically easier to select the `<form>` tag itself for serialization:

```
1 | $('form').submit(function() {
2 |     console.log($('this').serializeArray());
3 |     return false;
4 | });
```

This produces the following data structure (provided that the browser supports `console.log`):

```
1 | [
2 |   {
3 |     name: "a",
4 |     value: "1"
5 |   },
6 |   {
7 |     name: "b",
8 |     value: "2"
9 |   },
10 |  {
11 |    name: "c",
12 |    value: "3"
13 |  },
14 |  {
15 |    name: "d",
16 |    value: "4"
```

```
17     },
18     {
19         name: "e",
20         value: "5"
21     }
22 ]
```

## Example:

*Get the values from a form iterate through them, and append them to a results display.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     body, select { font-size:14px; }
6     form { margin:5px; }
7     p { color:red; margin:5px; }
8     b { color:blue; }
9   </style>
10  <script src="http://code.jquery.com/jquery-
11 </head>
12 <body>
13   <p><b>Results:</b> <span id="results"></span>
14
15   <form>
16     <select name="single">
17       <option>Single</option>
18       <option>Single2</option>
19
20     </select>
21     <select name="multiple" multiple="multiple"
22       <option selected="selected">Multiple</option>
23       <option>Multiple2</option>
24
25       <option selected="selected">Multiple3</option>
26     </select><br/>
27     <input type="checkbox" name="check" value="
28
29     <label for="ch1">check1</label>
30     <input type="checkbox" name="check" value="
```

```
31
32     <label for="ch2">check2</label>
33     <input type="radio" name="radio" value="radio1"
34
35     <label for="r1">radio1</label>
36     <input type="radio" name="radio" value="radio2"
37
38     <label for="r2">radio2</label>
39 </form>
40 <script>
41     function showValues() {
42         var fields = $(":input").serializeArray();
43         $("#results").empty();
44         jQuery.each(fields, function(i, field){
45             $("#results").append(field.value + " ");
46         });
47     }
48
49     $(":checkbox, :radio").click(showValues);
50     $("select").change(showValues);
51     showValues();
52 </script>
53
54 </body>
55 </html>
```

Demo

**A new version of this book is available!**

---

# .show()

Categories: [Effects](#) > [Basics](#)

`.show()`

Returns: [jQuery](#)

**Description:** *Display the matched elements.*

`.show()`

version added: [1.0](#)

This method does not accept any arguments.

`.show( [duration ] [, complete ] )`

version added: [1.0](#)

**duration** (default: `400`)

Type: [Number](#) or [String](#)

A string or number determining how long the animation will run.

**complete**

Type: [Function\(\)](#)

A function to call once the animation is complete.

`.show( options )`

version added: [1.0](#)

**options**

Type: [PlainObject](#)

A map of additional options to pass to the method.

**duration** (default: `400`)

Type: [Number](#) or [String](#)

A string or number determining how long the animation will run.

**easing** (default: `swing`)

Type: [String](#)

A string indicating which easing function to use for the transition.

**queue** (default: `true`)

Type: [Boolean](#)

A Boolean indicating whether to place the animation in the effects queue. If false, the animation will begin immediately.

**As of jQuery 1.7**, the queue option can also accept a string, in which case the animation is added to the queue represented by that string.

## specialEasing

Type: [PlainObject](#)

A map of one or more of the CSS properties defined by the properties argument and their corresponding easing functions. **(version added: 1.4)**

---

## step

Type: [Function](#)( [Number](#) now, [Tween](#) tween )

A function to be called for each animated property of each animated element. This function provides an opportunity to modify the Tween object to change the value of the property before it is set.

---

## progress

Type: [Function](#)( [Promise](#) animation, [Number](#) progress, [Number](#) remainingMs )

A function to be called after each step of the animation, only once per animated element regardless of the number of animated properties. **(version added: 1.8)**

---

## complete

Type: [Function](#)()

A function to call once the animation is complete.

---

## done

Type: [Function](#)( [Promise](#) animation, [Boolean](#) jumpedToEnd )

A function to be called when the animation completes (its Promise object is resolved). **(version added: 1.8)**

---

## fail

Type: [Function](#)( [Promise](#) animation, [Boolean](#) jumpedToEnd )

A function to be called when the animation fails to complete (its Promise object is rejected). **(version added: 1.8)**

---

## always

Type: [Function](#)( [Promise](#) animation, [Boolean](#) jumpedToEnd )

A function to be called when the animation completes or stops without completing (its Promise object is either resolved or rejected). **(version added: 1.8)**

## `.show( [duration ] [, easing ] [, complete ] )`

version added: [1.4.3](#)

**duration** (default: `400`)

Type: [Number](#) or [String](#)

A string or number determining how long the animation will run.

**easing** (default: `swing`)

Type: [String](#)

A string indicating which easing function to use for the transition.

**complete**

Type: [Function\(\)](#)

A function to call once the animation is complete.

With no parameters, the `.show()` method is the simplest way to display an element:

```
1 | $(' .target' ).show();
```

The matched elements will be revealed immediately, with no animation. This is roughly equivalent to calling `.css('display', 'block')`, except that the `display` property is restored to whatever it was initially. If an element has a `display` value of `inline`, then is hidden and shown, it will once again be displayed `inline`.

**Note:** If using `!important` in your styles, such as `display: none !important`, it is necessary to override the style using `.css('display', 'block !important')` should you wish for `.show()` to function correctly.

When a duration, a plain object, or a "complete" function is provided, `.show()` becomes an animation method. The `.show()` method animates the width, height, and opacity of the matched elements simultaneously.

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings `'fast'` and `'slow'` can be supplied to indicate durations of `200` and `600` milliseconds, respectively.

As of jQuery 1.4.3, an optional string naming an easing function may be used. Easing functions specify the speed at which the animation progresses at different points within the animation. The only easing implementations in the jQuery library are the default, called `swing`, and one that progresses at a constant pace, called `linear`. More easing functions are available with the use of plug-ins, most notably the [jQuery UI suite](#).

If supplied, the callback is fired once the animation is complete. This can be useful for stringing different animations together in sequence. The callback is not sent any arguments, but `this` is set to the DOM element being animated. If multiple elements are animated, it is important to note that the callback is executed once per matched element, not once for the animation as a whole.

We can animate any element, such as a simple image:

```
1 | <div id="clickme">
2 |   Click here
3 | </div>
4 | 
2  <html>
3  <head>
4    <style>
5    p { background:yellow; }
6  </style>
7    <script src="http://code.jquery.com/jquery-1.4.2.js"></script>
8  </head>
9  <body>
10   <button>Show it</button>
11     <p style="display: none">Hello 2</p>
12   <script>
13     $("button").click(function () {
14       $("p").show("slow");
15     });
16   </script>
17
18 </body>
19 </html>
```

### Demo

**Example:** *Show the first div followed by each next adjacent sibling div in order with a 200ms animation. Each animation starts when the previous sibling div's animation ends.*

```
1  <!DOCTYPE html>
```

```

2 <html>
3 <head>
4   <style>
5     div { background:#def3ca; margin:3px; width:80
6     display:none; float:left; text-align:center;
7   </style>
8   <script src="http://code.jquery.com/jquery-latest.js"
9 </head>
10 <body>
11
12   <button id="showr">Show</button>
13   <button id="hidr">Hide</button>
14   <div>Hello 3,</div>
15
16   <div>how</div>
17   <div>are</div>
18   <div>you?</div>
19 <script>
20   $("#showr").click(function () {
21     $("div").first().show("fast", function showNext
22     $(this).next("div").show("fast", showNext);
23   });
24 });
25
26   $("#hidr").click(function () {
27     $("div").hide(1000);
28   });
29 </script>
30
31 </body>
32 </html>

```

Demo

**Example:** Show all span and input elements with an animation. Change the text once the animation is done.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     span { display:none; }
6     div { display:none; }
7     p { font-weight:bold; background-color:#fcd
8   </style>
9   <script src="http://code.jquery.com/jquery-
10 </head>
11 <body>
12
13 <button>Do it!</button>
14 <span>Are you sure? (type 'yes' if you are) <
15 <div>
16   <form>
17     <input type="text" value="as;ldkfjalsdf".
18   </form>
19 </div>
20 <p style="display:none;">I'm hidden...</p>
21
22 <script>
23 function doIt() {
24   $("span,div").show("slow");
25 }
26 /* can pass in function name */
27 $("button").click(doIt);
28
29 $("form").submit(function () {
30   if ($("#input").val() == "yes") {
31     $("#p").show(4000, function () {
32       $(this).text("Ok, DONE! (now showing)")
33     });
34   }
35   $("span,div").hide("fast");
36   /* to stop the submit */
```

```
37     return false;  
38 });  
39 </script>  
40  
41 </body>  
42 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# .siblings()

Categories: [Traversing](#) > [Tree Traversal](#)

`.siblings( [selector ] )`

Returns: [jQuery](#)

**Description:** *Get the siblings of each element in the set of matched elements, optionally filtered by a selector.*

`.siblings( [selector ] )`

version added: [1.0](#)

**selector**

Type: [Selector](#)

A string containing a selector expression to match elements against.

Given a jQuery object that represents a set of DOM elements, the `.siblings()` method allows us to search through the siblings of these elements in the DOM tree and construct a new jQuery object from the matching elements.

The method optionally accepts a selector expression of the same type that we can pass to the `$( )` function. If the selector is supplied, the elements will be filtered by testing whether they match it.

Consider a page with a simple list on it:

```
1 | <ul>
2 |   <li>list item 1</li>
3 |   <li>list item 2</li>
4 |   <li class="third-item">list item 3</li>
5 |   <li>list item 4</li>
6 |   <li>list item 5</li>
7 | </ul>
```

If we begin at the third item, we can find its siblings:

```
1 | $('li.third-item').siblings().css('background'
```

The result of this call is a red background behind items 1, 2, 4, and 5. Since we do not supply a selector expression, all of the siblings are part of the object. If we had supplied one, only the matching items among these four would be included.

The original element is not included among the siblings, which is important to remember when we wish to find all elements at a particular level of the DOM tree.

## Examples:

**Example:** Find the unique siblings of all yellow li elements in the 3 lists (including other yellow li elements if appropriate).

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5      ul { float:left; margin:5px; font-size:16px
6      p { color:blue; margin:10px 20px; font-size
7          font-weight:bolder; }
8      .hilite { background:yellow; }
9  </style>
10  <script src="http://code.jquery.com/jquery-
11 </head>
12 <body>
13   <ul>
14     <li>One</li>
15     <li>Two</li>
16     <li class="hilite">Three</li>
17     <li>Four</li>
18   </ul>
19
20   <ul>
21     <li>Five</li>
22     <li>Six</li>
23     <li>Seven</li>
24   </ul>
25   <ul>
26     <li>Eight</li>
27     <li class="hilite">Nine</li>
28     <li>Ten</li>
29     <li class="hilite">Eleven</li>
```

```

30 </ul>
31 <p>Unique siblings: <b></b></p>
32 <script>
33 var len = $(".hilite").siblings()
34         .css("color", "red")
35         .length;
36 $("b").text(len);
37 </script>
38
39 </body>
40 </html>

```

Demo

**Example:** *Find all siblings with a class "selected" of each div.*

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-
5 </head>
6 <body>
7   <div><span>Hello</span></div>
8
9   <p class="selected">Hello Again</p>
10  <p>And Again</p>
11  <script>$("p").siblings(".selected").css("bac
12
13 </body>
14 </html>

```

Demo

POWERED BY HERONOTE

A new version of this book is available!

---

# .size()

Categories: [Miscellaneous](#) > [DOM Element Methods](#)

`.size()`

Returns: [Number](#)  
version deprecated: [1.8](#)

**Description:** *Return the number of elements in the jQuery object.*

`.size()`

version added: [1.0](#)

This method does not accept any arguments.

The `.size()` method is deprecated as of jQuery 1.8. Use the `.length` property instead.

The `.size()` method is functionally equivalent to the `.length` property; however, **the `.length` property is preferred** because it does not have the overhead of a function call.

Given a simple unordered list on the page:

```
1 | <ul>
2 |   <li>foo</li>
3 |   <li>bar</li>
4 | </ul>
```

Both `.size()` and `.length` identify the number of items:

```
1 | alert( "Size: " + $("li").size() );
2 | alert( "Size: " + $("li").length );
```

This results in two alerts:

Size: 2

Size: 2



## Example:

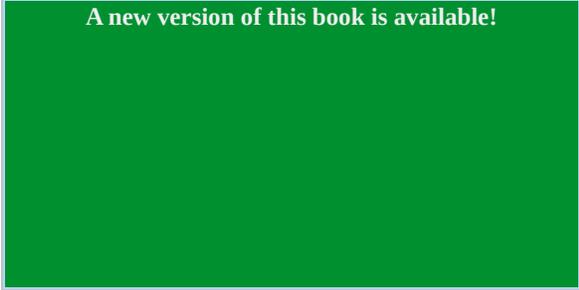
Count the divs. Click to add more.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     body { cursor:pointer; min-height: 100px; }
6     div { width:50px; height:30px; margin:5px;
7           float:left; background:blue; }
8     span { color:red; }
9   </style>
10  <script src="http://code.jquery.com/jquery-
11 </head>
12 <body>
13
14 <span></span>
15 <div></div>
16
17 <script>
18 $(document.body)
19 .click(function() {
20   $(this).append( $("<div>") );
21   var n = $("div").size();
22   $("span").text("There are " + n + " divs. C
23 })
24 // trigger the click to start
25 .click();
26 </script>
27
28 </body>
29 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

# .slice()

Categories: [Traversing](#) > [Filtering](#)

`.slice( start [, end ] )`

Returns: [jQuery](#)

**Description:** *Reduce the set of matched elements to a subset specified by a range of indices.*

`.slice( start [, end ] )`

version added: [1.1.4](#)

**start**

Type: [Integer](#)

An integer indicating the 0-based position at which the elements begin to be selected. If negative, it indicates an offset from the end of the set.

**end**

Type: [Integer](#)

An integer indicating the 0-based position at which the elements stop being selected. If negative, it indicates an offset from the end of the set. If omitted, the range continues until the end of the set.

Given a jQuery object that represents a set of DOM elements, the `.slice()` method constructs a new jQuery object containing a subset of the elements specified by the `start` and, optionally, `end` argument. The supplied `start` index identifies the position of one of the elements in the set; if `end` is omitted, all elements after this one will be included in the result.

Consider a page with a simple list on it:

```
1 | <ul>
2 |   <li>list item 1</li>
3 |   <li>list item 2</li>
4 |   <li>list item 3</li>
5 |   <li>list item 4</li>
6 |   <li>list item 5</li>
7 | </ul>
```

We can apply this method to the set of list items:

```
1 | $('li').slice(2).css('background-color', 'red')
```

The result of this call is a red background for items 3, 4, and 5. Note that the supplied index is zero-based, and refers to the position of elements within the jQuery object, not within the DOM tree.

The end parameter allows us to limit the selected range even further. For example:

```
1 | $('li').slice(2, 4).css('background-color', 'red')
```

Now only items 3 and 4 are selected. The index is once again zero-based; the range extends up to but not including the specified index.

### Negative Indices

The jQuery `.slice()` method is patterned after the JavaScript `.slice()` method for arrays. One of the features that it mimics is the ability for negative numbers to be passed as either the `start` or `end` parameter. If a negative number is provided, this indicates a position starting from the end of the set, rather than the beginning. For example:

```
1 | $('li').slice(-2, -1).css('background-color', 'red')
```

This time only list item 4 is turned red, since it is the only item in the range between two from the end (`-2`) and one from the end (`-1`).

## Examples:

**Example:** *Turns divs yellow based on a random slice.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5      div { width:40px; height:40px; margin:10px;
6            border:2px solid blue; }
7      span { color:red; font-weight:bold; }
8      button { margin:5px; }
9    </style>
10   <script src="http://code.jquery.com/jquery-
11 </head>
12 <body>
13   <p><button>Turn slice yellow</button>
14   <span>Click the button!</span></p>
15   <div></div>
16   <div></div>
17
18   <div></div>
19   <div></div>
20   <div></div>
21   <div></div>
22   <div></div>
23
24   <div></div>
25   <div></div>
26 <script>
27 function colorEm() {
28   var $div = $("div");
29   var start = Math.floor(Math.random() *
30                       $div.length);
31   var end = Math.floor(Math.random() *
```

```

32         ($div.length - start)) +
33         start + 1;
34     if (end == $div.length) end = undefined;
35     $div.css("background", "");
36     if (end)
37         $div.slice(start, end).css("background", "yellow")
38     else
39         $div.slice(start).css("background", "yellow")
40
41     $("span").text('$("div").slice(' + start +
42         (end ? ', ' + end : '') +
43         ').css("background", "yellow");'
44 }
45
46 $("button").click(colorEm);
47 </script>
48
49 </body>
50 </html>

```

## Demo

**Example** · Selects all paragraphs, then slices the selection to include only the first element.

```
1 | $("p").slice(0, 1).wrapInner("<b></b>");
```

**Example** · Selects all paragraphs, then slices the selection to include only the first and second element.

```
1 | $("p").slice(0, 2).wrapInner("<b></b>");
```

**Example** · Selects all paragraphs, then slices the selection to include only the second element.

```
1 | $("p").slice(1, 2).wrapInner("<b></b>");
```

**Example** · *Selects all paragraphs, then slices the selection to include only the second and third element.*

```
1 | $("p").slice(1).wrapInner("<b></b>");
```

**Example** · *Selects all paragraphs, then slices the selection to include only the third element.*

```
1 | $("p").slice(-1).wrapInner("<b></b>");
```

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# .slideDown()

Categories: [Effects](#) > [Sliding](#)

`.slideDown( [duration ] [, complete ] )`

Returns: [jQuery](#)

**Description:** *Display the matched elements with a sliding motion.*

`.slideDown( [duration ] [, complete ] )` version added: [1.0](#)

**duration** (default: `400`)

Type: [Number](#) or [String](#)

A string or number determining how long the animation will run.

**complete**

Type: [Function\(\)](#)

A function to call once the animation is complete.

`.slideDown( options )`

version added: [1.0](#)

**options**

Type: [PlainObject](#)

A map of additional options to pass to the method.

**duration** (default: `400`)

Type: [Number](#) or [String](#)

A string or number determining how long the animation will run.

**easing** (default: `swing`)

Type: [String](#)

A string indicating which easing function to use for the transition.

**queue** (default: `true`)

Type: [Boolean](#)

A Boolean indicating whether to place the animation in the effects queue. If false, the animation will begin immediately.

**As of jQuery 1.7**, the queue option can also accept a string, in which case the animation is added to the queue represented by that string.

**specialEasing**

Type: [PlainObject](#)

A map of one or more of the CSS properties defined by the properties argument and their corresponding easing functions. **(version added: 1.4)**

### **step**

Type: [Function](#)( [Number](#) now, [Tween](#) tween )

A function to be called for each animated property of each animated element. This function provides an opportunity to modify the Tween object to change the value of the property before it is set.

### **progress**

Type: [Function](#)( [Promise](#) animation, [Number](#) progress, [Number](#) remainingMs )

A function to be called after each step of the animation, only once per animated element regardless of the number of animated properties. **(version added: 1.8)**

### **complete**

Type: [Function](#)()

A function to call once the animation is complete.

### **done**

Type: [Function](#)( [Promise](#) animation, [Boolean](#) jumpedToEnd )

A function to be called when the animation completes (its Promise object is resolved). **(version added: 1.8)**

### **fail**

Type: [Function](#)( [Promise](#) animation, [Boolean](#) jumpedToEnd )

A function to be called when the animation fails to complete (its Promise object is rejected). **(version added: 1.8)**

### **always**

Type: [Function](#)( [Promise](#) animation, [Boolean](#) jumpedToEnd )

A function to be called when the animation completes or stops without completing (its Promise object is either resolved or rejected). **(version added: 1.8)**

**.slideDown( [duration ] [, easing ] [, version added: 1.4.3 complete ] )**

**duration** (default: `400`)

Type: [Number](#) or [String](#)

A string or number determining how long the animation will run.

**easing** (default: `swing`)

Type: [String](#)

A string indicating which easing function to use for the transition.

**complete**

Type: [Function\(\)](#)

A function to call once the animation is complete.

The `.slideDown()` method animates the height of the matched elements. This causes lower parts of the page to slide down, making way for the revealed items.

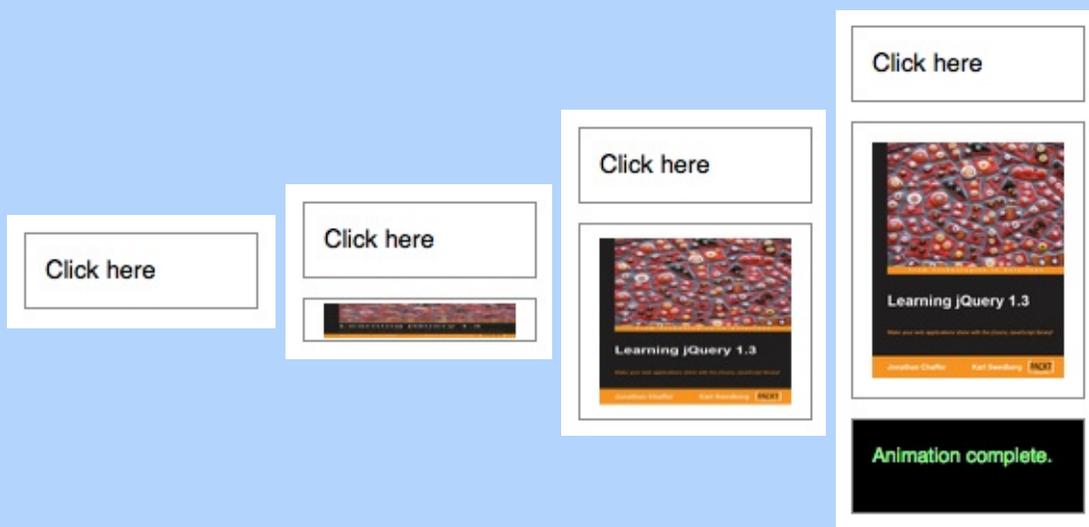
Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings `'fast'` and `'slow'` can be supplied to indicate durations of `200` and `600` milliseconds, respectively. If any other string is supplied, or if the `duration` parameter is omitted, the default duration of `400` milliseconds is used.

We can animate any element, such as a simple image:

```
1 | <div id="clickme">
2 |   Click here
3 | </div>
4 | 
```

With the element initially hidden, we can show it slowly:

```
1 | $('#clickme').click(function() {
2 |   $('#book').slideDown('slow', function() {
3 |     // Animation complete.
4 |   });
```



## Easing

As of **jQuery 1.4.3**, an optional string naming an easing function may be used. Easing functions specify the speed at which the animation progresses at different points within the animation. The only easing implementations in the jQuery library are the default, called `swing`, and one that progresses at a constant pace, called `linear`. More easing functions are available with the use of plug-ins, most notably the [jQuery UI suite](#).

## Callback Function

If supplied, the callback is fired once the animation is complete. This can be useful for stringing different animations together in sequence. The callback is not sent any arguments, but `this` is set to the DOM element being animated. If multiple elements are animated, it is important to note that the callback is executed once per matched element, not once for the animation as a whole.

As of **jQuery 1.6**, the `.promise()` method can be used in conjunction with the `deferred.done()` method to execute a single callback for the animation as a whole when *all* matching elements have completed their animations ( See the [example for .promise\(\)](#) ).

## Additional Notes:

All jQuery effects, including `.slideDown()`, can be turned off globally by setting `jQuery.fx.off = true`, which effectively sets the duration to 0. For more information, see [jQuery.fx.off](#).

If `.slideDown()` is called on an unordered list (`<ul>`) and its `<li>` elements have position (relative, absolute, or fixed), the effect may not work properly in IE6 through at least IE9 unless the `<ul>` has "layout." To remedy the problem, add the `position: relative;` and `zoom: 1;` CSS declarations to the `ul`.

## Examples:

**Example:** *Animates all divs to slide down and show themselves over 600 milliseconds.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5  div { background:#de9a44; margin:3px; width:80px;
6  height:40px; display:none; float:left; }
7  </style>
8    <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
9  </head>
10 <body>
11   Click me!
12 <div></div>
13 <div></div>
14 <div></div>
15 <script>
16 $(document.body).click(function () {
17   if ($("#div:first").is(":hidden")) {
18     $("#div").slideDown("slow");
19   } else {
20     $("#div").hide();
21   }
22 });
23
24 </script>
25
26 </body>
27 </html>
```

## Demo

**Example:** Animates all inputs to slide down, completing the animation within 1000 milliseconds. Once the animation is done, the input look is changed especially if it is the middle input which gets the focus.

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5  div { background:#cfd; margin:3px; width:50px
6  text-align:center; float:left; cursor:pointer
7  border:2px outset black; font-weight:bolder;
8  input { display:none; width:120px; float:left
9  margin:10px; }
10 </style>
11   <script src="http://code.jquery.com/jquery-
12 </head>
13 <body>
14   <div>Push!</div>
15   <input type="text" />
16   <input type="text" class="middle" />
17
18   <input type="text" />
19   <script>
20   $("div").click(function () {
21   $(this).css({ borderStyle:"inset", cursor:"wa.
22   $("input").slideDown(1000,function(){
23   $(this).css("border", "2px red inset")
24   .filter(".middle")
25   .css("background", "yellow")
26   .focus();
27   $("div").css("visibility", "hidden");
28   });
29   });
30
```

```
31 </script>
32
33 </body>
34 </html>
```

Demo

POWERED BY **HERONOTE**

A new version of this book is available!

---

# **.slideToggle()**

Categories: [Effects](#) > [Sliding](#)

`.slideToggle( [duration ] [, complete ] )`

Returns: [jQuery](#)

**Description:** *Display or hide the matched elements with a sliding motion.*

`.slideToggle( [duration ] [, complete ] )` version added: [1.0](#)  
)

**duration** (default: `400`)

Type: [Number](#) or [String](#)

A string or number determining how long the animation will run.

**complete**

Type: [Function\(\)](#)

A function to call once the animation is complete.

`.slideToggle( options )`

version added: [1.0](#)

**options**

Type: [PlainObject](#)

A map of additional options to pass to the method.

**duration** (default: `400`)

Type: [Number](#) or [String](#)

A string or number determining how long the animation will run.

**easing** (default: `swing`)

Type: [String](#)

A string indicating which easing function to use for the transition.

**queue** (default: `true`)

Type: [Boolean](#)

A Boolean indicating whether to place the animation in the effects queue. If false, the animation will begin immediately.

**As of jQuery 1.7**, the queue option can also accept a string, in which case the animation is added to the queue represented by that string.

**specialEasing**

Type: [PlainObject](#)

A map of one or more of the CSS properties defined by the properties argument and their corresponding easing functions. **(version added: 1.4)**

---

### step

Type: [Function](#)( [Number](#) now, [Tween](#) tween )

A function to be called for each animated property of each animated element. This function provides an opportunity to modify the Tween object to change the value of the property before it is set.

---

### progress

Type: [Function](#)( [Promise](#) animation, [Number](#) progress, [Number](#) remainingMs )

A function to be called after each step of the animation, only once per animated element regardless of the number of animated properties. **(version added: 1.8)**

---

### complete

Type: [Function](#)()

A function to call once the animation is complete.

---

### done

Type: [Function](#)( [Promise](#) animation, [Boolean](#) jumpedToEnd )

A function to be called when the animation completes (its Promise object is resolved). **(version added: 1.8)**

---

### fail

Type: [Function](#)( [Promise](#) animation, [Boolean](#) jumpedToEnd )

A function to be called when the animation fails to complete (its Promise object is rejected). **(version added: 1.8)**

---

### always

Type: [Function](#)( [Promise](#) animation, [Boolean](#) jumpedToEnd )

A function to be called when the animation completes or stops without completing (its Promise object is either resolved or rejected). **(version added: 1.8)**

---

**.slideToggle( [duration ] [, easing ]** version added: 1.4.3

## [, complete ] )

**duration** (default: 400)

Type: [Number](#) or [String](#)

A string or number determining how long the animation will run.

**easing** (default: swing)

Type: [String](#)

A string indicating which easing function to use for the transition.

**complete**

Type: [Function\(\)](#)

A function to call once the animation is complete.

The `.slideToggle()` method animates the height of the matched elements. This causes lower parts of the page to slide up or down, appearing to reveal or conceal the items. If the element is initially displayed, it will be hidden; if hidden, it will be shown. The `display` property is saved and restored as needed. If an element has a `display` value of `inline`, then is hidden and shown, it will once again be displayed `inline`. When the height reaches 0 after a hiding animation, the `display` style property is set to `none` to ensure that the element no longer affects the layout of the page.

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings `'fast'` and `'slow'` can be supplied to indicate durations of 200 and 600 milliseconds, respectively.

We can animate any element, such as a simple image:

```
1 | <div id="clickme">
2 |   Click here
3 | </div>
4 | 
```

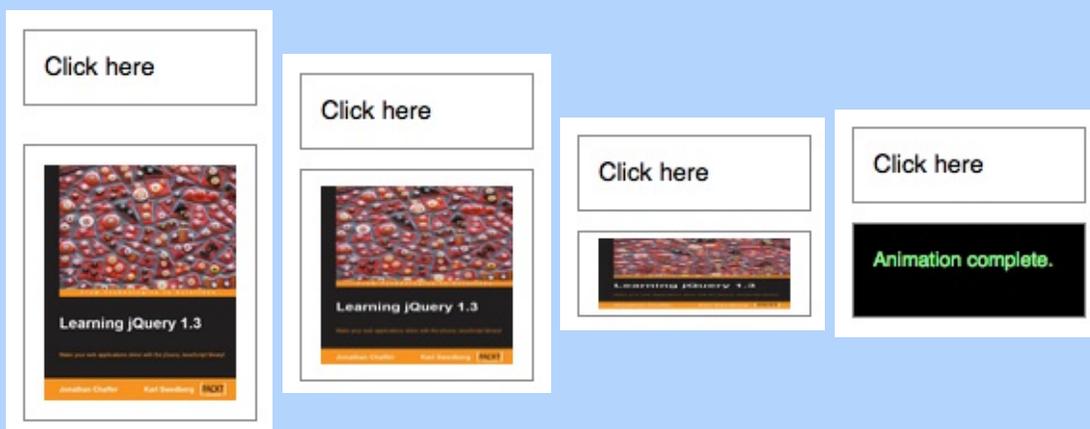
We will cause `.slideToggle()` to be called when another element is clicked:

```

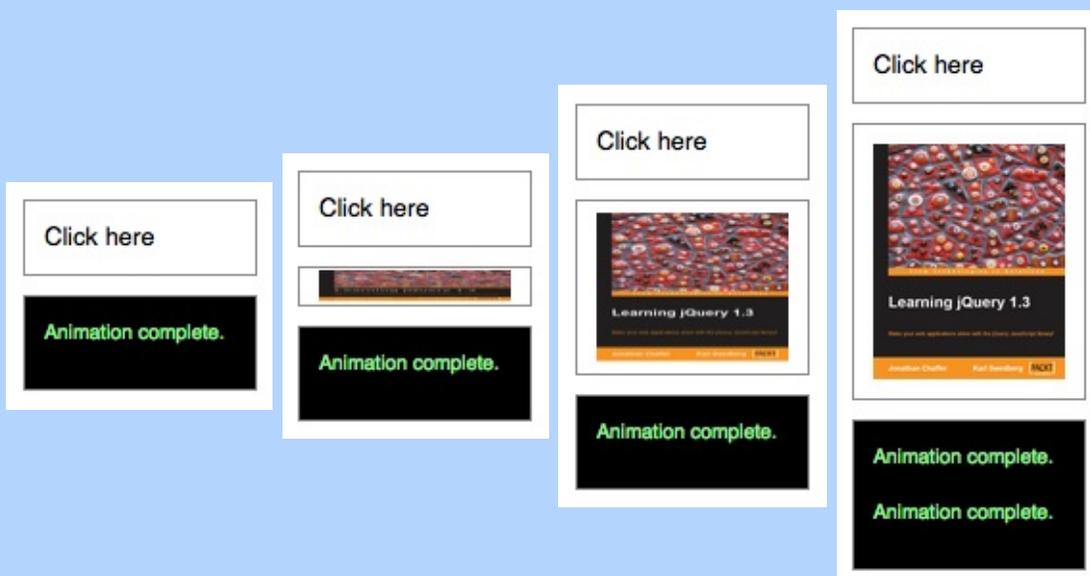
1 | $('#clickme').click(function() {
2 |     $('#book').slideToggle('slow', function() {
3 |         // Animation complete.
4 |     });
5 | });

```

With the element initially shown, we can hide it slowly with the first click:



A second click will show the element once again:



**Easing**

**As of jQuery 1.4.3**, an optional string naming an easing function may be used. Easing functions specify the speed at which the animation progresses at different points within the animation. The only easing implementations in the jQuery library are the default, called `swing`, and one that progresses at a constant pace, called `linear`. More easing functions are available with the use of plug-ins, most notably the [jQuery UI suite](#).

## Callback Function

If supplied, the callback is fired once the animation is complete. This can be useful for stringing different animations together in sequence. The callback is not sent any arguments, but `this` is set to the DOM element being animated. If multiple elements are animated, it is important to note that the callback is executed once per matched element, not once for the animation as a whole.

**As of jQuery 1.6**, the `.promise()` method can be used in conjunction with the `deferred.done()` method to execute a single callback for the animation as a whole when *all* matching elements have completed their animations ( See the [example for .promise\(\)](#) ).

## Additional Notes:

All jQuery effects, including `.slideToggle()`, can be turned off globally by setting `jQuery.fx.off = true`, which effectively sets the duration to 0. For more information, see [jQuery.fx.off](#).

If `.slideDown()` is called on an unordered list (`<ul>`) and its `<li>` elements have position (relative, absolute, or fixed), the effect may not work properly in IE6 through at least IE9 unless the `<ul>` has "layout." To remedy the problem, add the `position: relative;` and `zoom: 1;` CSS declarations to the `ul`.

## Examples:

**Example:** *Animates all paragraphs to slide in or down, completing the animation within 600 milliseconds.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     p { width:400px; }
6   </style>
7   <script src="http://code.jquery.com/jquery-...
8 </head>
9 <body>
10  <button>Toggle</button>
11
12  <p>
13    This is the paragraph to end all paragraphs
14    should feel lucky to have seen some
15    your life. Congratulations!
16  </p>
17  <script>
18    $("button").click(function () {
19      $("p").slideToggle("slow");
20    });
21  </script>
22
23 </body>
24 </html>
```

Demo

**Example:** *Animates divs between dividers with a toggle that makes some appear and some disappear.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     div { background:#b977d1; margin:3px; width:60
6           height:60px; float:left; }
7     div.still { background:#345; width:5px; }
8     div.hider { display:none; }
9     span { color:red; }
10    p { clear: left; }</style>
11    <script src="http://code.jquery.com/jquery-latest.js"
12  </head>
13  <body>
14    <div></div>
15    <div class="still"></div>
16    <div style="display:none;"
17  </div><div class="still"></div>
18  <div></div>
19  <div class="still"></div>
20  <div class="hider"></div>
21  <div class="still"></div>
22  <div class="hider"></div>
23  <div class="still"></div>
24  <div></div>
25  <p><button id="aa">Toggle</button> There have been
26  <script>
27    $(" #aa").click(function () {
28      $("div:not(.still)").slideToggle("slow", function
29        var n = parseInt($("span").text(), 10);
30        $("span").text(n + 1);
31      });
32    });
33
34  </script>
35
36  </body>
```

37 | `</html>`

Demo

POWERED BY **HERONOTE**

A new version of this book is available!

---

# `.slideUp()`

Categories: [Effects](#) > [Sliding](#)

`.slideUp( [duration ] [, complete ] )`

Returns: [jQuery](#)

**Description:** *Hide the matched elements with a sliding motion.*

`.slideUp( [duration ] [, complete ] )`

version added: [1.0](#)

**duration** (default: `400`)

Type: [Number](#) or [String](#)

A string or number determining how long the animation will run.

**complete**

Type: [Function\(\)](#)

A function to call once the animation is complete.

`.slideUp( options )`

version added: [1.0](#)

**options**

Type: [PlainObject](#)

A map of additional options to pass to the method.

**duration** (default: `400`)

Type: [Number](#) or [String](#)

A string or number determining how long the animation will run.

**easing** (default: `swing`)

Type: [String](#)

A string indicating which easing function to use for the transition.

**queue** (default: `true`)

Type: [Boolean](#)

A Boolean indicating whether to place the animation in the effects queue. If false, the animation will begin immediately.

**As of jQuery 1.7**, the queue option can also accept a string, in which case the animation is added to the queue represented by that string.

**specialEasing**

Type: [PlainObject](#)

A map of one or more of the CSS properties defined by the properties argument and their corresponding easing functions. **(version added: 1.4)**

### step

Type: [Function](#)( [Number](#) now, [Tween](#) tween )

A function to be called for each animated property of each animated element. This function provides an opportunity to modify the Tween object to change the value of the property before it is set.

### progress

Type: [Function](#)( [Promise](#) animation, [Number](#) progress, [Number](#) remainingMs )

A function to be called after each step of the animation, only once per animated element regardless of the number of animated properties. **(version added: 1.8)**

### complete

Type: [Function](#)()

A function to call once the animation is complete.

### done

Type: [Function](#)( [Promise](#) animation, [Boolean](#) jumpedToEnd )

A function to be called when the animation completes (its Promise object is resolved). **(version added: 1.8)**

### fail

Type: [Function](#)( [Promise](#) animation, [Boolean](#) jumpedToEnd )

A function to be called when the animation fails to complete (its Promise object is rejected). **(version added: 1.8)**

### always

Type: [Function](#)( [Promise](#) animation, [Boolean](#) jumpedToEnd )

A function to be called when the animation completes or stops without completing (its Promise object is either resolved or rejected). **(version added: 1.8)**

**.slideUp( [duration ] [, easing ] [, complete ] )**

version added: [1.4.3](#)

**duration** (default: `400`)

Type: `Number` or `String`

A string or number determining how long the animation will run.

**easing** (default: `swing`)

Type: `String`

A string indicating which easing function to use for the transition.

**complete**

Type: `Function()`

A function to call once the animation is complete.

The `.slideUp()` method animates the height of the matched elements. This causes lower parts of the page to slide up, appearing to conceal the items. Once the height reaches 0 (or, if set, to whatever the CSS min-height property is), the `display` style property is set to `none` to ensure that the element no longer affects the layout of the page.

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings `'fast'` and `'slow'` can be supplied to indicate durations of `200` and `600` milliseconds, respectively. If any other string is supplied, or if the `duration` parameter is omitted, the default duration of `400` milliseconds is used.

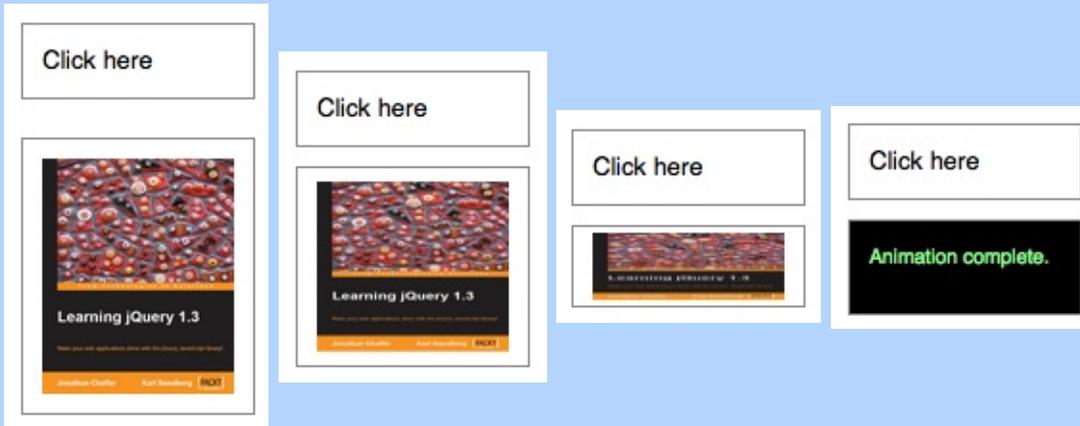
We can animate any element, such as a simple image:

```
1 | <div id="clickme">
2 |   Click here
3 | </div>
4 | 
```

With the element initially shown, we can hide it slowly:

```
1 | $('#clickme').click(function() {
2 |   $('#book').slideUp('slow', function() {
```

```
3 | // Animation complete.  
4 | });  
5 | });
```



## Easing

As of **jQuery 1.4.3**, an optional string naming an easing function may be used. Easing functions specify the speed at which the animation progresses at different points within the animation. The only easing implementations in the jQuery library are the default, called `swing`, and one that progresses at a constant pace, called `linear`. More easing functions are available with the use of plug-ins, most notably the [jQuery UI suite](#).

## Callback Function

If supplied, the callback is fired once the animation is complete. This can be useful for stringing different animations together in sequence. The callback is not sent any arguments, but `this` is set to the DOM element being animated. If multiple elements are animated, it is important to note that the callback is executed once per matched element, not once for the animation as a whole.

As of **jQuery 1.6**, the `.promise()` method can be used in conjunction with the `deferred.done()` method to execute a single callback for the animation as a whole when *all* matching elements have completed their animations ( See the [example for .promise\(\)](#) ).

## Additional Notes:

All jQuery effects, including `.slideUp()`, can be turned off globally by setting `jQuery.fx.off = true`, which effectively sets the duration to 0. For more information, see [jQuery.fx.off](#).

If `.slideDown()` is called on an unordered list (`<ul>`) and its `<li>` elements have position (relative, absolute, or fixed), the effect may not work properly in IE6 through at least IE9 unless the `<ul>` has "layout." To remedy the problem, add the `position: relative;` and `zoom: 1;` CSS declarations to the `ul`.

## Examples:

**Example:** *Animates all divs to slide up, completing the animation within 400 milliseconds.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5      div { background:#3d9a44; margin:3px; width
6        height:40px; float:left; }
7    </style>
8    <script src="http://code.jquery.com/jquery-
9  </head>
10 <body>
11   Click me!
12   <div></div>
13   <div></div>
14   <div></div>
15   <div></div>
16
17   <div></div>
18 <script>
19   $(document.body).click(function () {
20     if ($("#div:first").is(":hidden")) {
21       $("#div").show("slow");
22     } else {
23       $("#div").slideUp();
24     }
25   });
26
27 </script>
28
29 </body>
30 </html>
```

## Demo

**Example:** *Animates the parent paragraph to slide up, completing the animation within 200 milliseconds. Once the animation is done, it displays an alert.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5      div { margin:2px; }
6    </style>
7    <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
8  </head>
9  <body>
10   <div>
11     <button>Hide One</button>
12     <input type="text" value="One" />
13
14   </div>
15   <div>
16     <button>Hide Two</button>
17     <input type="text" value="Two" />
18
19   </div>
20   <div>
21     <button>Hide Three</button>
22     <input type="text" value="Three" />
23
24   </div>
25   <div id="msg"></div>
26 </body>
27 <script>
28   $( "button" ).click( function () {
29     $( this ).parent().slideUp( "slow", function () {
30       alert( $( "#msg" ).text( $( "button", this ).text() ) );
31     } );
32   } );
33 </script>
34 </html>
```

```
30     });  
31     });  
32  
33     </script>  
34  
35     </body>  
36     </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# .stop()

Categories: [Effects](#) > [Custom](#)

`.stop( [clearQueue ] [, jumpToEnd ] )`

Returns: [jQuery](#)

**Description:** *Stop the currently-running animation on the matched elements.*

`.stop( [clearQueue ] [, jumpToEnd ] )` version added: [1.2](#)

**clearQueue**

Type: [Boolean](#)

A Boolean indicating whether to remove queued animation as well. Defaults to `false`.

**jumpToEnd**

Type: [Boolean](#)

A Boolean indicating whether to complete the current animation immediately. Defaults to `false`.

`.stop( [queue ] [, clearQueue ] [, jumpToEnd ] )`

version added: [1.7](#)

**queue**

Type: [String](#)

The name of the queue in which to stop animations.

**clearQueue**

Type: [Boolean](#)

A Boolean indicating whether to remove queued animation as well. Defaults to `false`.

**jumpToEnd**

Type: [Boolean](#)

A Boolean indicating whether to complete the current animation immediately. Defaults to `false`.

When `.stop()` is called on an element, the currently-running animation (if any) is immediately stopped. If, for instance, an element is being hidden with `.slideUp()` when `.stop()` is called, the element will now still be displayed, but will be a fraction of its previous height. Callback functions are not called.

If more than one animation method is called on the same element, the later animations are placed in the effects queue for the element. These animations will not begin until the first one completes. When `.stop()` is called, the next animation in the queue begins immediately. If the `clearQueue` parameter is provided with a value of `true`, then the rest of the animations in the queue are removed and never run.

If the `jumpToEnd` argument is provided with a value of `true`, the current animation stops, but the element is immediately given its target values for each CSS property. In our above `.slideUp()` example, the element would be immediately hidden. The callback function is then immediately called, if provided.

**As of jQuery 1.7**, if the first argument is provided as a string, only the animations in the queue represented by that string will be stopped.

The usefulness of the `.stop()` method is evident when we need to animate an element on `mouseenter` and `mouseleave`:

```
1 | <div id="hoverme">
2 |   Hover me
3 |   
```

We can create a nice fade effect without the common problem of multiple queued animations by adding `.stop(true, true)` to the chain:

```
1 | $('#hoverme-stop-2').hover(function() {
2 |   $(this).find('img').stop(true, true).fadeOut(
3 | }, function() {
4 |   $(this).find('img').stop(true, true).fadeIn(
5 | });
```



## Toggling Animations

As of jQuery 1.7, stopping a toggled animation prematurely with `.stop()` will trigger jQuery's internal effects tracking. In previous versions, calling the `.stop()` method before a toggled animation was completed would cause the animation to lose track of its state (if `jumpToEnd` was false). Any subsequent animations would start at a new "half-way" state, sometimes resulting in the element disappearing. To observe the new behavior, see the final example below.

*Animations may be stopped globally by setting the property `$.fx.off` to `true`. When this is done, all animation methods will immediately set elements to their final state when called, rather than displaying an effect.*

## Examples:

**Example:** Click the Go button once to start the animation then click the STOP button to stop it where it's currently positioned. Another option is to click several buttons to queue them up and see that stop just kills the currently playing one.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>div {
5     position: absolute;
6     background-color: #abc;
7     left: 0px;
8     top: 30px;
9     width: 60px;
10    height: 60px;
11    margin: 5px;
12  }
13 </style>
14   <script src="http://code.jquery.com/jquery-
15 </head>
16 <body>
17   <button id="go">Go</button>
18   <button id="stop">STOP!</button>
19   <button id="back">Back</button>
20   <div class="block"></div>
21   <script>
22     /* Start animation */
23     $("#go").click(function(){
24     $(".block").animate({left: '+=100px'}, 2000);
25     });
26
27     /* Stop animation when button is clicked */
```

```

28  $("#stop").click(function(){
29  $(".block").stop();
30  });
31
32  /* Start animation in the opposite direction */
33  $("#back").click(function(){
34  $(".block").animate({left: '-=100px'}, 2000);
35  });
36
37  </script>
38
39  </body>
40  </html>

```

## Demo

**Example:** Click the slideToggle button to start the animation then click again before the animation is completed. The animation will toggle the other direction from the saved starting point.

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>.block {
5    background-color: #abc;
6    border: 2px solid black;
7    width: 200px;
8    height: 80px;
9    margin: 10px;
10   }
11  </style>
12  <script src="http://code.jquery.com/jquery-
13  </head>
14  <body>
15    <button id="toggle">slideToggle</button>

```

```
16 <div class="block"></div>
17 <script>
18 var $block = $('#block');
19 /* Toggle a sliding animation animation */
20 $('#toggle').on('click', function() {
21     $block.stop().slideToggle(1000);
22 });
23 </script>
24
25 </body>
26 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# .submit()

Categories: [Events](#) > [Form Events](#) | [Forms](#)

`.submit( handler(eventObject) )`

Returns: [jQuery](#)

**Description:** *Bind an event handler to the "submit" JavaScript event, or trigger that event on an element.*

`.submit( handler(eventObject) )`

version added: [1.0](#)

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute each time the event is triggered.

`.submit( [ eventData ],  
handler(eventObject) )`

version added: [1.4.3](#)

**eventData**

Type: [PlainObject](#)

An object containing data that will be passed to the event handler.

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute each time the event is triggered.

`.submit()`

version added: [1.0](#)

This method does not accept any arguments.

This method is a shortcut for `.on('submit', handler)` in the first variation, and `.trigger('submit')` in the third.

The `submit` event is sent to an element when the user is attempting to submit a form. It can only be attached to `<form>` elements. Forms can be submitted either by clicking an explicit `<input type="submit">`, `<input type="image">`, or `<button type="submit">`, or by pressing Enter when certain form elements have focus.

*Depending on the browser, the Enter key*

*may only cause a form submission if the form has exactly one text field, or only when there is a submit button present. The interface should not rely on a particular behavior for this key unless the issue is forced by observing the keypress event for presses of the Enter key.*

For example, consider the HTML:

```
1 | <form id="target" action="destination.html">
2 |   <input type="text" value="Hello there" />
3 |   <input type="submit" value="Go" />
4 | </form>
5 | <div id="other">
6 |   Trigger the handler
7 | </div>
```

The event handler can be bound to the form:

```
1 | $('#target').submit(function() {
2 |   alert('Handler for .submit() called.');
```

```
3 |   return false;
4 | });
```

Now when the form is submitted, the message is alerted. This happens prior to the actual submission, so we can cancel the submit action by calling `.preventDefault()` on the event object or by returning `false` from our handler. We can trigger the event manually when another element is clicked:

```
1 | $('#other').click(function() {  
2 |     $('#target').submit();  
3 | });
```

After this code executes, clicks on Trigger the handler will also display the message. In addition, the default `submit` action on the form will be fired, so the form will be submitted.

The JavaScript `submit` event does not bubble in Internet Explorer. However, scripts that rely on event delegation with the `submit` event will work consistently across browsers as of jQuery 1.4, which has normalized the event's behavior.

## Additional Notes:

Forms and their child elements should not use input names or ids that conflict with properties of a form, such as `submit`, `length`, or `method`. Name conflicts can cause confusing failures. For a complete list of rules and to check your markup for these problems, see [DOMLint](#).

## Examples:

**Example:** *If you'd like to prevent forms from being submitted unless a flag variable is set, try:*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5
6   p { margin:0; color:blue; }
7   div,p { margin-left:10px; }
8   span { color:red; }
9 </style>
10  <script src="http://code.jquery.com/jquery-
11 </head>
12 <body>
13   <p>Type 'correct' to validate.</p>
14   <form action="javascript:alert('success!');
15     <div>
16       <input type="text" />
17
18       <input type="submit" />
19     </div>
20   </form>
21   <span></span>
22 <script>
23
24 $( "form" ).submit( function() {
25   if ( $( "input:first" ).val() == "correct" ) {
26     $( "span" ).text( "Validated..." ).show();
27     return true;
28   }
29   $( "span" ).text( "Not valid!" ).show().fadeOut
30   return false;
```

```
31 | });  
32 | </script>  
33 |  
34 | </body>  
35 | </html>
```

## Demo

**Example:** *If you'd like to prevent forms from being submitted unless a flag variable is set, try:*

```
1 | $("form").submit( function () {  
2 |     return this.some_flag_variable;  
3 | } );
```

**Example:** *To trigger the submit event on the first form on the page, try:*

```
1 | $("form:first").submit();
```

POWERED BYHERONOTE

A new version of this book is available!

---

# :submit Selector

Categories: [Selectors](#) > [Form](#) | [Selectors](#) > [jQuery Extensions](#)

## submit selector

**Description:** *Selects all elements of type submit.*

**jQuery( ":submit" )**

version added: [1.0](#)

The `:submit` selector typically applies to button or input elements. Note that some browsers treat `<button>` element as `type="default"` implicitly while others (such as Internet Explorer) do not.

### Additional Notes:

Because `:submit` is a jQuery extension and not part of the CSS specification, queries using `:submit` cannot take advantage of the performance boost provided by the native DOM `querySelectorAll()` method. For better performance in modern browsers, use `[type="submit"]` instead.

## Example:

*Find all submit elements that are descendants of a td element.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     textarea { height:45px; }
6   </style>
7   <script src="http://code.jquery.com/jquery"
8 </head>
9 <body>
10
11 <form>
12 <table id="exampleTable" border="1" cellpadding="5"
13   <tr>
14     <th>
15       Element Type
16     </th>
17     <th>
18       Element
19     </th>
20   </tr>
21   <tr>
22     <td>
23       <input type="button" value="Input Button" />
24     </td>
25   </tr>
26   <tr>
27     <td>
28       <input type="checkbox" />
29     </td>
30   </tr>
```

```
31 <tr>
32   <td>
33     <input type="file" />
34   </td>
35 </tr>
36 <tr>
37   <td>
38     <input type="hidden" />
39   </td>
40 </tr>
41 <tr>
42   <td>
43     <input type="image" />
44   </td>
45 </tr>
46 <tr>
47   <td>
48     <input type="password" />
49   </td>
50 </tr>
51 <tr>
52   <td>
53     <input type="radio" />
54   </td>
55 </tr>
56 <tr>
57   <td>
58     <input type="reset" />
59   </td>
60 </tr>
61 <tr>
62   <td>
63     <input type="submit" />
64   </td>
65 </tr>
66 <tr>
67 <tr>
```

```
68     <td>
69         <input type="text" />
70     </td>
71 </tr>
72 <tr>
73     <td>
74         <select><option>Option</option></select>
75     </td>
76 </tr>
77 <tr>
78     <td>
79         <textarea></textarea>
80     </td>
81 </tr>
82 <tr>
83     <td>
84         <button>Button</button>
85     </td>
86 </tr>
87 <tr>
88     <td>
89         <button type="submit">Button type="submit"
90     </td>
91 </tr>
92
93 </table>
94 </form>
95 <div id="result"></div>
96
97 <script>
98 var submitEl = $("td :submit")
99     .parent('td')
100     .css({background: "yellow", border: "3px red solid"}
101     .end());
102
103 $('#result').text('jQuery matched ' + submitEl.length + '
104
```

```
105 // so it won't submit
106 $("form").submit(function () { return false; });
107
108 // Extra JS to make the HTML easier to edit (None of t
109 $('#exampleTable').find('td').each(function(i, el) {
110     var inputEl = $(el).children(),
111         inputType = inputEl.attr('type') ? ' type="'
112         $(el).before('<td>' + inputEl[0].nodeName + inputTY
113     })
114 </script>
115
116 </body>
117 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# :target Selector

Categories: [Selectors](#) > [Basic Filter](#)

## target selector

**Description:** *Selects the target element indicated by the fragment identifier of the document's URI.*

**jQuery( ":target" )**

version added: [1.9](#)

If the document's URI contains a fragment identifier, or hash, then the `:target` selector will match the element with an ID that matches the identifier. For example, given a document with a URI of `http://example.com/#foo`, `$( "p:target" )` will select the `<p id="foo">` element.

Further discussion of this usage can be found in the [W3C CSS specification](#).

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# .text()

Categories: [Manipulation](#) > [DOM Insertion, Inside](#)

Get the combined text contents of each element in the set of matched elements, including their descendants, or set the text contents of the matched elements.

## Contents:

[.text\(\)](#)

`.text()`

[.text\( textString \)](#)

`.text( textString )`

`.text( function(index, text) )`

`.text()`

Returns: `String`

**Description:** *Get the combined text contents of each element in the set of matched elements, including their descendants.*

`.text()`

version added: `1.0`

This method does not accept any arguments.

Unlike the `.html()` method, `.text()` can be used in both XML and HTML documents. The result of the `.text()` method is a string containing the combined text of all matched elements. (Due to variations in the HTML parsers in different browsers, the text returned may vary in newlines and other white space.) Consider the following HTML:

```
1 <div class="demo-container">
2   <div class="demo-box">Demonstration Box<
3   <ul>
4   <li>list item 1</li>
5   <li>list <strong>item</strong> 2</li>
6   </ul>
7   </div>
```

The code `$('#div.demo-container').text()` would produce the following result:

```
Demonstration Box list item 1 list item 2
```

The `.text()` method cannot be used on form inputs or scripts. To set or get the text value of `input` or `textarea` elements, use the `.val()` method. To get the value of a script element, use the `.html()` method.

As of jQuery 1.4, the `.text()` method returns the value of text and

CDATA nodes as well as element nodes.

## Example:

*Find the text in the first paragraph (stripping out the html) then set the html of the last paragraph to show it is just text (the red bold is gone).*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5      p { color:blue; margin:8px; }
6      b { color:red; }
7    </style>
8    <script src="http://code.jquery.com/jquery-
9  </head>
10 <body>
11   <p><b>Test</b> Paragraph.</p>
12   <p></p>
13   <script>
14   var str = $("p:first").text();
15   $("p:last").html(str);
16   </script>
17
18 </body>
19 </html>
```

Demo

`.text( textString )`

Returns: [jQuery](#)

**Description:** *Set the content of each element in the set of matched elements to the specified text.*

`.text( textString )`

version added: [1.0](#)

**textString**

Type: [String](#)

A string of text to set as the content of each matched element.

`.text( function(index, text) )`

version added: [1.4](#)

**function(index, text)**

Type: [Function\(\)](#)

A function returning the text content to set. Receives the index position of the element in the set and the old text value as arguments.

Unlike the `.html()` method, `.text()` can be used in both XML and HTML documents.

We need to be aware that this method escapes the string provided as necessary so that it will render correctly in HTML. To do so, it calls the DOM method `.createTextNode()`, which replaces special characters with their HTML entity equivalents (such as `&lt;` for `<`). Consider the following HTML:

```
1 <div class="demo-container">
2   <div class="demo-box">Demonstration Box</div>
3   <ul>
4     <li>list item 1</li>
5     <li>list <strong>item</strong> 2</li>
6   </ul>
7 </div>
```

The code `$('#div.demo-container').text('<p>This is a test.</p>');` will produce the following DOM output:

```
1 | <div class="demo-container">
2 | &lt;p&gt;This is a test.&lt;/p&gt;
3 | </div>
```

It will appear on a rendered page as though the tags were exposed, like this:

```
1 | <p>This is a test</p>
```

The `.text()` method cannot be used on input elements. For input field text, use the `.val()` method.

As of jQuery 1.4, the `.text()` method allows us to set the text content by passing in a function.

```
1 | $('ul li').text(function(index) {
2 |     return 'item number ' + (index + 1);
3 | });
```

Given an unordered list with three `<li>` elements, this example will produce the following DOM output:

```
1 | <ul>
2 |     <li>item number 1</li>
3 |     <li>item number 2</li>
4 |     <li>item number 3</li>
5 | </ul>
```

## Example:

*Add text to the paragraph (notice the bold tag is escaped).*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5
6   p { color:blue; margin:8px; }
7 </style>
8   <script src="http://code.jquery.com/jquery-
9 </head>
10 <body>
11   <p>Test Paragraph.</p>
12   <script>$("p").text("<b>Some</b> new text.");
13
14 </body>
15 </html>
```

Demo

POWERED BY HERONOTE

A new version of this book is available!

---

# :text Selector

Categories: [Selectors](#) > [Form](#) | [Selectors](#) > [jQuery Extensions](#)

## text selector

**Description:** *Selects all elements of type text.*

**jQuery( ":text" )**

version added: [1.0](#)

`$('.text')` allows us to select all `<input type="text">` elements. As with other pseudo-class selectors (those that begin with a ":"), it is recommended to precede it with a tag name or some other selector; otherwise, the universal selector ("\*") is implied. In other words, the bare `$('.text')` is equivalent to `$('.*:text')`, so `$('.input:text')` should be used instead.

**Note:** As of jQuery 1.5.2, `:text` selects `input` elements that have no specified `type` attribute (in which case `type="text"` is implied).

This difference in behavior between `$('.text')` and `$('.[type=text]')`, can be seen below:

```
1 | $('<input>').is('[type=text]'); // false
2 | $('<input>').is(':text'); // true
```

## Additional Notes:

Because `:text` is a jQuery extension and not part of the CSS specification, queries using `:text` cannot take advantage of the performance boost provided by the native DOM `querySelectorAll()` method. For better performance in modern browsers, use `['type="text"]` instead.

## Example:

*Finds all text inputs.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5      textarea { height:25px; }
6    </style>
7    <script src="http://code.jquery.com/jquery-
8  </head>
9  <body>
10   <form>
11     <input type="button" value="Input Button"
12     <input type="checkbox" />
13
14     <input type="file" />
15     <input type="hidden" />
16     <input type="image" />
17
18     <input type="password" />
19     <input type="radio" />
20     <input type="reset" />
21
22     <input type="submit" />
23     <input type="text" />
24     <select><option>Option</option></select>
25
26     <textarea></textarea>
27     <button>Button</button>
28   </form>
29   <div>
30   </div>
31 </script>
```

```
32 var input = $("form input:text").css({background:
33 $("div").text("For this type jQuery found " + input.length
34 .css("color", "red");
35 $("form").submit(function () { return false; });
36 </script>
37
38 </body>
39 </html>
```

Demo

POWERED BYHERONOTE

A new version of this book is available!

---

# .toArray()

Categories: [Miscellaneous](#) > [DOM Element Methods](#)

`.toArray()`

Returns: `Array`

**Description:** *Retrieve all the DOM elements contained in the jQuery set, as an array.*

`.toArray()`

version added: [1.4](#)

This method does not accept any arguments.

`.toArray()` returns all of the elements in the jQuery set:

```
1 | alert($('li').toArray());
```

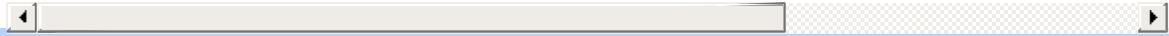
All of the matched DOM nodes are returned by this call, contained in a standard array:

[<li id="foo">, <li id="bar">]

## Example:

*Selects all divs in the document and returns the DOM Elements as an Array then uses the built-in reverse-method to reverse that array.*

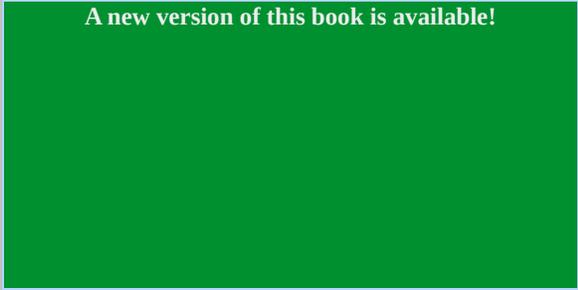
```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5      span { color:red; }
6    </style>
7    <script src="http://code.jquery.com/jquery-
8  </head>
9  <body>
10   Reversed - <span></span>
11
12   <div>One</div>
13   <div>Two</div>
14   <div>Three</div>
15  <script>
16  function disp(divs) {
17    var a = [];
18    for (var i = 0; i < divs.length; i++) {
19      a.push(divs[i].innerHTML);
20    }
21    $("span").text(a.join(" "));
22  }
23
24  disp( $("div").toArray().reverse() );
25  </script>
26
27  </body>
28  </html>
```



Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

# .toggle()

Categories: [Effects](#) > [Basics](#)

`.toggle( [duration ] [, complete ] )`

Returns: [jQuery](#)

**Description:** *Display or hide the matched elements.*

`.toggle( [duration ] [, complete ] )`

version added: [1.0](#)

**duration** (default: `400`)

Type: [Number](#) or [String](#)

A string or number determining how long the animation will run.

**complete**

Type: [Function\(\)](#)

A function to call once the animation is complete.

`.toggle( options )`

version added: [1.0](#)

**options**

Type: [PlainObject](#)

A map of additional options to pass to the method.

**duration** (default: `400`)

Type: [Number](#) or [String](#)

A string or number determining how long the animation will run.

**easing** (default: `swing`)

Type: [String](#)

A string indicating which easing function to use for the transition.

**queue** (default: `true`)

Type: [Boolean](#)

A Boolean indicating whether to place the animation in the effects queue. If false, the animation will begin immediately.

**As of jQuery 1.7**, the queue option can also accept a string, in which case the animation is added to the queue represented by that string.

**specialEasing**

Type: [PlainObject](#)

A map of one or more of the CSS properties defined by the

properties argument and their corresponding easing functions. **(version added: 1.4)**

### step

Type: [Function](#)( [Number](#) now, [Tween](#) tween )

A function to be called for each animated property of each animated element. This function provides an opportunity to modify the Tween object to change the value of the property before it is set.

### progress

Type: [Function](#)( [Promise](#) animation, [Number](#) progress, [Number](#) remainingMs )

A function to be called after each step of the animation, only once per animated element regardless of the number of animated properties. **(version added: 1.8)**

### complete

Type: [Function](#)()

A function to call once the animation is complete.

### done

Type: [Function](#)( [Promise](#) animation, [Boolean](#) jumpedToEnd )

A function to be called when the animation completes (its Promise object is resolved). **(version added: 1.8)**

### fail

Type: [Function](#)( [Promise](#) animation, [Boolean](#) jumpedToEnd )

A function to be called when the animation fails to complete (its Promise object is rejected). **(version added: 1.8)**

### always

Type: [Function](#)( [Promise](#) animation, [Boolean](#) jumpedToEnd )

A function to be called when the animation completes or stops without completing (its Promise object is either resolved or rejected). **(version added: 1.8)**

**.toggle( [duration ] [, easing ] [, complete ] )**

version added: [1.4.3](#)

**duration** (default: `400`)

Type: [Number](#) or [String](#)

A string or number determining how long the animation will run.

**easing** (default: `swing`)

Type: [String](#)

A string indicating which easing function to use for the transition.

**complete**

Type: [Function\(\)](#)

A function to call once the animation is complete.

**.toggle( showOrHide )**

version added: [1.3](#)

**showOrHide**

Type: [Boolean](#)

A Boolean indicating whether to show or hide the elements.

*Note: The event handling suite also has a method named [.toggle\(\)](#). Which one is fired depends on the set of arguments passed.*

With no parameters, the `.toggle()` method simply toggles the visibility of elements:

```
1 | $(' .target' ).toggle();
```

The matched elements will be revealed or hidden immediately, with no animation, by changing the CSS `display` property. If the element is initially displayed, it will be hidden; if hidden, it will be shown. The `display` property is saved and restored as needed. If an element has a `display` value of `inline`, then is hidden and shown, it will once again be displayed `inline`.

When a duration, a plain object, or a *single* "complete" function is

provided, `.toggle()` becomes an animation method. The `.toggle()` method animates the width, height, and opacity of the matched elements simultaneously. When these properties reach 0 after a hiding animation, the `display` style property is set to `none` to ensure that the element no longer affects the layout of the page.

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings `'fast'` and `'slow'` can be supplied to indicate durations of `200` and `600` milliseconds, respectively.

As of jQuery 1.4.3, an optional string naming an easing function may be used. Easing functions specify the speed at which the animation progresses at different points within the animation. The only easing implementations in the jQuery library are the default, called `swing`, and one that progresses at a constant pace, called `linear`. More easing functions are available with the use of plug-ins, most notably the [jQuery UI suite](#).

If supplied, the callback is fired once the animation is complete. This can be useful for stringing different animations together in sequence. The callback is not sent any arguments, but `this` is set to the DOM element being animated. If multiple elements are animated, it is important to note that the callback is executed once per matched element, not once for the animation as a whole.

We can animate any element, such as a simple image:

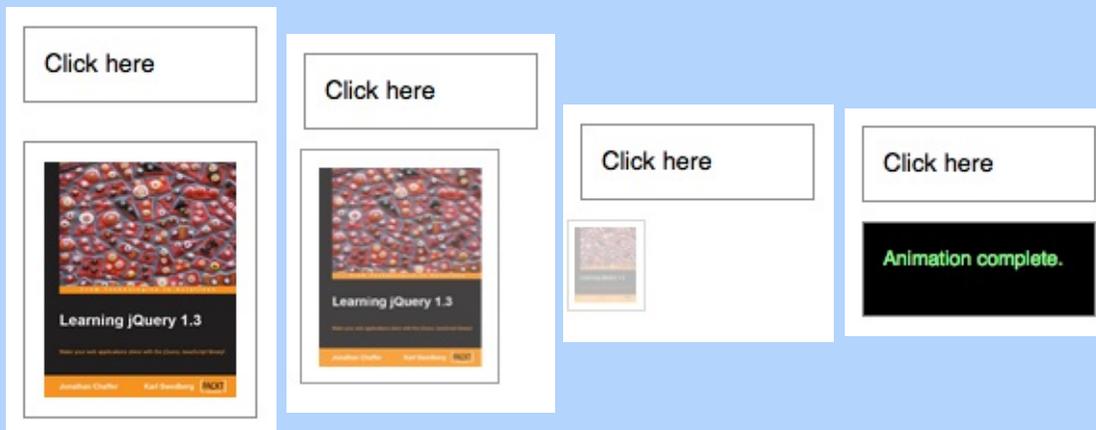
```
1 | <div id="clickme">
2 |   Click here
3 | </div>
4 | 
```

We will cause `.toggle()` to be called when another element is clicked:

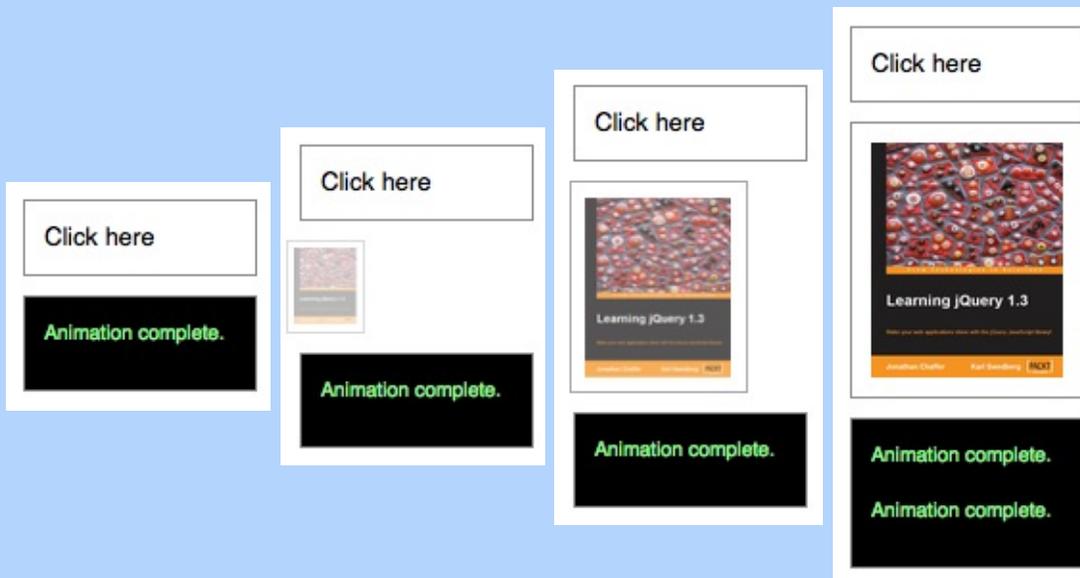
```
1 | $('#clickme').click(function() {
2 |   $('#book').toggle('slow', function() {
```

```
3 | // Animation complete.  
4 | });  
5 | });
```

With the element initially shown, we can hide it slowly with the first click:



A second click will show the element once again:



The second version of the method accepts a Boolean parameter. If this parameter is `true`, then the matched elements are shown; if `false`, the elements are hidden. In essence, the statement:

```
1 | $('#foo').toggle(showOrHide);
```

is equivalent to:

```
1 | if ( showOrHide == true ) {  
2 |     $( '#foo' ).show();  
3 | } else if ( showOrHide == false ) {  
4 |     $( '#foo' ).hide();  
5 | }
```

## Additional Notes:

All jQuery effects, including `.toggle()`, can be turned off globally by setting `jquery.fx.off = true`, which effectively sets the duration to 0. For more information, see [jQuery.fx.off](#).

## Examples:

**Example:** *Toggles all paragraphs.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-1.11.0.min.js"></script>
5 </head>
6 <body>
7   <button>Toggle</button>
8   <p>Hello</p>
9   <p style="display: none">Good Bye</p>
10  <script>
11
12    $("button").click(function () {
13      $("p").toggle();
14    });
15  </script>
16
17 </body>
18 </html>
```

### Demo

**Example:** *Animates all paragraphs to be shown if they are hidden and hidden if they are visible, completing the animation within 600 milliseconds.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     p { background:#dad;
```

```

6 font-weight:bold;
7 font-size:16px; }
8 </style>
9 <script src="http://code.jquery.com/jquery-latest.js"
10 </head>
11 <body>
12 <button>Toggle 'em</button>
13
14 <p>Hiya</p>
15 <p>Such interesting text, eh?</p>
16 <script>
17 $("button").click(function () {
18 $("p").toggle("slow");
19 });
20 </script>
21
22 </body>
23 </html>

```

Demo

**Example** · Shows all paragraphs, then hides them all, back and forth.

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4 <script src="http://code.jquery.com/jquery-
5 </head>
6 <body>
7 <button>Toggle</button>
8 <p>Hello</p>
9 <p style="display: none">Good Bye</p>
10 <script>
11 var flip = 0;
12 $("button").click(function () {

```

```
13 | $("p").toggle( flip++ % 2 == 0 );  
14 | });  
15 | </script>  
16 |  
17 | </body>  
18 | </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# .toggle()

Categories: [Deprecated](#) > [Deprecated 1.8](#) | [Events](#) > [Mouse Events](#)

`.toggle( handler(eventObject), handler(eventObject) [, handler(eventObject) ] )` Returns: [jQuery](#)  
version deprecated: [1.8](#), removed: [1.9](#)

**Description:** *Bind two or more handlers to the matched elements, to be executed on alternate clicks.*

`.toggle( handler(eventObject), handler(eventObject) [, handler(eventObject) ] )` version added: [1.0](#)

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute every even time the element is clicked.

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute every odd time the element is clicked.

**handler(eventObject)**

Type: [Function\(\)](#)

Additional handlers to cycle through after clicks.

*Note: jQuery also provides an animation method named [.toggle\(\)](#) that toggles the visibility of elements. Whether the animation or the event method is fired depends on the set of arguments passed.*

The `.toggle()` method binds a handler for the `click` event, so the rules outlined for the triggering of `click` apply here as well.

```
1 | For example, consider the HTML:  
2 | <div id="target">
```

```
3 | Click here
4 | </div>
```

Event handlers can then be bound to the `<div>`:

```
1 | $('#target').toggle(function() {
2 |     alert('First handler for .toggle() called.')
3 | }, function() {
4 |     alert('Second handler for .toggle() called.')
5 | });
```

As the element is clicked repeatedly, the messages alternate:

First handler for .toggle() called. Second handler for .toggle() called.  
First handler for .toggle() called.  
Second handler for .toggle() called.  
First handler for .toggle() called.

If more than two handlers are provided, `.toggle()` will cycle among all of them. For example, if there are three handlers, then the first handler will be called on the first click, the fourth click, the seventh click, and so on.

The `.toggle()` method is provided for convenience. It is relatively straightforward to implement the same behavior by hand, and this can be necessary if the assumptions built into `.toggle()` prove limiting. For example, `.toggle()` is not guaranteed to work correctly if applied twice to the same element. Since `.toggle()` internally uses a `click` handler to do its work, we must unbind `click` to remove a behavior attached with `.toggle()`, so other `click` handlers can be caught in the crossfire. The implementation also calls `.preventDefault()` on the event, so links will not be followed and buttons will not be clicked if `.toggle()` has been called on the element.

## Example:

*Toggle a style on table cells. (Not recommended. Use `.toggleClass()` instead.):*

```
1 | $("td").toggle(  
2 |   function () {  
3 |     $(this).addClass("selected");  
4 |   },  
5 |   function () {  
6 |     $(this).removeClass("selected");  
7 |   }  
8 | );
```

POWERED BY HERONOTE

A new version of this book is available!

---

# .toggleClass()

Categories: [Attributes](#) | [Manipulation](#) > [Class Attribute](#) | [CSS](#)

`.toggleClass( className )`

Returns: [jQuery](#)

**Description:** *Add or remove one or more classes from each element in the set of matched elements, depending on either the class's presence or the value of the switch argument.*

**.toggleClass( className )**

version added: [1.0](#)

**className**

Type: [String](#)

One or more class names (separated by spaces) to be toggled for each element in the matched set.

**.toggleClass( className, switch )**

version added: [1.3](#)

**className**

Type: [String](#)

One or more class names (separated by spaces) to be toggled for each element in the matched set.

**switch**

Type: [Boolean](#)

A Boolean (not just truthy/falsy) value to determine whether the class should be added or removed.

**.toggleClass( [switch ] )**

version added: [1.4](#)

**switch**

Type: [Boolean](#)

A boolean value to determine whether the class should be added or removed.

**.toggleClass( function(index, class, switch) [, switch ] )**

version added: [1.4](#)

**function(index, class, switch)**

Type: [Function\(\)](#)

A function that returns class names to be toggled in the class attribute of each element in the matched set. Receives the

index position of the element in the set, the old class value, and the switch as arguments.

### switch

Type: [Boolean](#)

A boolean value to determine whether the class should be added or removed.

This method takes one or more class names as its parameter. In the first version, if an element in the matched set of elements already has the class, then it is removed; if an element does not have the class, then it is added. For example, we can apply `.toggleClass()` to a simple `<div>`:

```
1 | <div class="tumble">Some text.</div>
```

The first time we apply `$('#div.tumble').toggleClass('bounce')`, we get the following:

```
1 | <div class="tumble bounce">Some text.</div>
```

The second time we apply `$('#div.tumble').toggleClass('bounce')`, the `<div>` class is returned to the single `tumble` value:

```
1 | <div class="tumble">Some text.</div>
```

Applying `.toggleClass('bounce spin')` to the same `<div>` alternates between `<div class="tumble bounce spin">` and `<div class="tumble">`.

The second version of `.toggleClass()` uses the second parameter for determining whether the class should be added or removed. If this parameter's value is `true`, then the class is added; if `false`, the class is removed. In essence, the statement:

```
1 | $('#foo').toggleClass(className, addOrRemove);
```

is equivalent to:

```
1 | if (addOrRemove) {  
2 |     $('#foo').addClass(className);  
3 | }  
4 | else {  
5 |     $('#foo').removeClass(className);  
6 | }
```

As of jQuery 1.4, if no arguments are passed to `.toggleClass()`, all class names on the element the first time `.toggleClass()` is called will be toggled. Also as of jQuery 1.4, the class name to be toggled can be determined by passing in a function.

```
1 | $('#div.foo').toggleClass(function() {  
2 |     if ($(this).parent().is('.bar')) {  
3 |         return 'happy';  
4 |     } else {  
5 |         return 'sad';  
6 |     }  
7 | });
```

This example will toggle the `happy` class for `<div class="foo">` elements if their parent element has a class of `bar`; otherwise, it will toggle the `sad` class.

## Examples:

**Example:** Toggle the class 'highlight' when a paragraph is clicked.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5
6   p { margin: 4px; font-size:16px; font-weight:
7     cursor:pointer; }
8   .blue { color:blue; }
9   .highlight { background:yellow; }
10 </style>
11 <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
12 </head>
13 <body>
14   <p class="blue">Click to toggle</p>
15   <p class="blue highlight">highlight</p>
16   <p class="blue">on these</p>
17   <p class="blue">paragraphs</p>
18 <script>
19   $("p").click(function () {
20     $(this).toggleClass("highlight");
21   });
22 </script>
23
24 </body>
25 </html>
```

Demo

**Example:** Add the "highlight" class to the clicked

*paragraph on every third click of that paragraph,  
remove it every first and second click.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     p { margin: 4px; font-size:16px; font-weight: bold;
6         cursor:pointer; }
7     .blue { color:blue; }
8     .highlight { background:red; }
9   </style>
10  <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
11 </head>
12 <body>
13   <p class="blue">Click to toggle (<span>clicks: 0</span>).
14   <p class="blue highlight">highlight (<span>clicks: 0</span>).
15   <p class="blue">on these (<span>clicks: 0</span>).
16   <p class="blue">on these (<span>clicks: 0</span>).
17   <p class="blue">paragraphs (<span>clicks: 0</span>).
18 <script>
19   var count = 0;
20   $("p").each(function() {
21     var $thisParagraph = $(this);
22     var count = 0;
23     $thisParagraph.click(function() {
24       count++;
25       $thisParagraph.find("span").text('clicks: ' + count);
26       $thisParagraph.toggleClass("highlight", count % 3 == 0);
27     });
28   });
29
30 </script>
31
32 </body>
33 </html>
```

## Demo

**Example:** Toggle the class name(s) indicated on the buttons for each div.

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5    .wrap > div { float: left; width: 100px; margin: 5px;
6                  padding-left: 3px; border: 1px solid #ccc; }
7    div.a { background-color: aqua; }
8    div.b { background-color: burlywood; }
9    div.c { background-color: cornsilk; }
10 </style>
11   <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
12 </head>
13 <body>
14
15 <div class="buttons">
16   <button>toggle</button>
17   <button class="a">toggle a</button>
18   <button class="a b">toggle a b</button>
19   <button class="a b c">toggle a b c</button>
20   <a href="#">reset</a>
21 </div>
22 <div class="wrap">
23   <div></div>
24   <div class="b"></div>
25   <div class="a b"></div>
26   <div class="a c"></div>
27 </div>
28
29 <script>
30 var cls = ['a', 'a b', 'a b c'];
31 var divs = $('div.wrap').children();
```

```
32 var appendClass = function() {
33     divs.append(function() {
34         return '<div>' + (this.className || 'none') +
35     });
36 };
37
38 appendClass();
39
40 $('button').on('click', function() {
41     var tc = this.className || undefined;
42     divs.toggleClass(tc);
43     appendClass();
44 });
45
46 $('a').on('click', function(event) {
47     event.preventDefault();
48     divs.empty().each(function(i) {
49         this.className = cls[i];
50     });
51     appendClass();
52 });
53 </script>
54
55 </body>
56 </html>
```

Demo

**A new version of this book is available!**

---

# .trigger()

Categories: [Events](#) > [Event Handler Attachment](#)

`.trigger( eventType [, extraParameters ] )` Returns: [jQuery](#)

**Description:** *Execute all handlers and behaviors attached to the matched elements for the given event type.*

`.trigger( eventType [, extraParameters ] )`

version added: [1.0](#)

**eventType**

Type: [String](#)

A string containing a JavaScript event type, such as `click` or `submit`.

**extraParameters**

Type: [Array](#) or [PlainObject](#)

Additional parameters to pass along to the event handler.

`.trigger( event )`

version added: [1.3](#)

**event**

Type: [Event](#)

A [jQuery.Event](#) object.

Any event handlers attached with `.on()` or one of its shortcut methods are triggered when the corresponding event occurs. They can be fired manually, however, with the `.trigger()` method. A call to `.trigger()` executes the handlers in the same order they would be if the event were triggered naturally by the user:

```
1 | $( '#foo' ).on( 'click', function() {  
2 |     alert( $( this ).text() );  
3 |     } );  
4 | $( '#foo' ).trigger( 'click' );
```

As of jQuery 1.3, `.trigger()`ed events bubble up the DOM tree; an event handler can stop the bubbling by returning `false` from the

handler or calling the `.stopPropagation()` method on the event object passed into the event. Although `.trigger()` simulates an event activation, complete with a synthesized event object, it does not perfectly replicate a naturally-occurring event.

To trigger handlers bound via jQuery without also triggering the native event, use `.triggerHandler()` instead.

When we define a custom event type using the `.on()` method, the second argument to `.trigger()` can become useful. For example, suppose we have bound a handler for the `custom` event to our element instead of the built-in `click` event as we did above:

```
1 | $('#foo').on('custom', function(event, param1,
2 |     alert(param1 + "\n" + param2);
3 | });
4 | $('#foo').trigger('custom', ['Custom', 'Event
```

The event object is always passed as the first parameter to an event handler, but if additional parameters are specified during a `.trigger()` call, these parameters will be passed along to the handler as well. To pass more than one parameter, use an array as shown here. As of jQuery 1.6.2, a single parameter can be passed without using an array.

Note the difference between the extra parameters we're passing here and the `eventData` parameter to the `.on()` method. Both are mechanisms for passing information to an event handler, but the `extraParameters` argument to `.trigger()` allows information to be determined at the time the event is triggered, while the `eventData` argument to `.on()` requires the information to be already computed at the time the handler is bound.

The `.trigger()` method can be used on jQuery collections that wrap plain JavaScript objects similar to a pub/sub mechanism; any event handlers bound to the object will be called when the event is triggered.

**Note:** For both plain objects and DOM objects, if a triggered event

name matches the name of a property on the object, jQuery will attempt to invoke the property as a method if no event handler calls `event.preventDefault()`. If this behavior is not desired, use `.triggerHandler()` instead.

## Examples:

**Example:** Clicks to button #2 also trigger a click for button #1.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5
6   button { margin:10px; }
7   div { color:blue; font-weight:bold; }
8   span { color:red; }
9 </style>
10   <script src="http://code.jquery.com/jquery-
11 </head>
12 <body>
13   <button>Button #1</button>
14   <button>Button #2</button>
15   <div><span>0</span> button #1 clicks.</div>
16
17   <div><span>0</span> button #2 clicks.</div>
18 <script>
19 $("button:first").click(function () {
20   update($("span:first"));
21 });
22 $("button:last").click(function () {
23   $("button:first").trigger('click');
24
25   update($("span:last"));
26 });
27
28 function update(j) {
29   var n = parseInt(j.text(), 10);
30   j.text(n + 1);
```

```
31 | }
32 | </script>
33 |
34 | </body>
35 | </html>
```

## Demo

**Example:** To submit the first form without using the `submit()` function, try:

```
1 | $("form:first").trigger("submit")
```

**Example:** To submit the first form without using the `submit()` function, try:

```
1 | var event = jQuery.Event("submit");
2 | $("form:first").trigger(event);
3 | if ( event.isDefaultPrevented() ) {
4 |   // Perform an action...
5 | }
```

**Example:** To pass arbitrary data to an event:

```
1 | $("p").click( function (event, a, b) {
2 |   // when a normal click fires, a and b are unde
3 |   // for a trigger like below a refers to "foo"
4 |
5 | } ).trigger("click", ["foo", "bar"]);
```

**Example:** To pass arbitrary data through an event object:

```
1 | var event = jQuery.Event("logged");
2 | event.user = "foo";
3 | event.pass = "bar";
4 | $("body").trigger(event);
```

**Example** · *Alternative way to pass data through an event object:*

```
1 | $("body").trigger({
2 |   type: "logged",
3 |   user: "foo",
4 |   pass: "bar"
5 | });
6 |
```

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# .triggerHandler()

Categories: [Events](#) > [Event Handler Attachment](#)

`.triggerHandler( eventType [, extraParameters ] )`

Returns: [Object](#)

**Description:** *Execute all handlers attached to an element for an event.*

`.triggerHandler( eventType [, extraParameters ] )`

version added: [1.2](#)

**eventType**

Type: [String](#)

A string containing a JavaScript event type, such as `click` or `submit`.

**extraParameters**

Type: [Array](#)

An array of additional parameters to pass along to the event handler.

The `.triggerHandler()` method behaves similarly to `.trigger()`, with the following exceptions:

The `.triggerHandler()` method does not cause the default behavior of an event to occur (such as a form submission).

While `.trigger()` will operate on all elements matched by the jQuery object, `.triggerHandler()` only affects the first matched element.

Events created with `.triggerHandler()` do not bubble up the DOM hierarchy; if they are not handled by the target element directly, they do nothing.

Instead of returning the jQuery object (to allow chaining), `.triggerHandler()` returns whatever value was returned by the last handler it caused to be executed. If no handlers are triggered, it returns `undefined`

For more information on this method, see the discussion for [.trigger\(\)](#).



## Example:

*If you called `triggerHandler()` on a focus event - the browser's default focus action would not be triggered, only the event handlers bound to the focus event.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-1.10.2.min.js"></script>
5 </head>
6 <body>
7   <button id="old">.trigger("focus")</button>
8   <button id="new">.triggerHandler("focus")</button>
9
10  <input type="text" value="To Be Focused"/>
11  <script>
12
13  $("#old").click(function(){
14    $("#input").trigger("focus");
15  });
16  $("#new").click(function(){
17    $("#input").triggerHandler("focus");
18  });
19  $("#input").focus(function(){
20    $("#<span>Focused!</span>").appendTo("body").fadeToggle(100);
21  });
22
23  </script>
24
25 </body>
26 </html>
```

POWERED BY **HERONOTE**

A new version of this book is available!

---

# .unbind()

Categories: [Events](#) > [Event Handler Attachment](#)

`.unbind( [eventType ] [, handler(eventObject) ] )`

Returns: [jQuery](#)

**Description:** *Remove a previously-attached event handler from the elements.*

`.unbind( [eventType ] [, handler(eventObject) ] )`

version added: [1.0](#)

**eventType**

Type: [String](#)

A string containing a JavaScript event type, such as `click` or `submit`.

**handler(eventObject)**

Type: [Function\(\)](#)

The function that is to be no longer executed.

`.unbind( eventType, false )`

version added: [1.4.3](#)

**eventType**

Type: [String](#)

A string containing a JavaScript event type, such as `click` or `submit`.

**false**

Type: [Boolean](#)

Unbinds the corresponding 'return false' function that was bound using `.bind( eventType, false )`.

`.unbind( event )`

version added: [1.0](#)

**event**

Type: [Object](#)

A JavaScript event object as passed to an event handler.

Event handlers attached with `.bind()` can be removed with `.unbind()`. (As of jQuery 1.7, the `.on()` and `.off()` methods are preferred to attach and remove event handlers on elements.) In the

simplest case, with no arguments, `.unbind()` removes all handlers attached to the elements:

```
1 | $('#foo').unbind();
```

This version removes the handlers regardless of type. To be more precise, we can pass an event type:

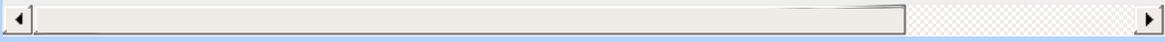
```
1 | $('#foo').unbind('click');
```

By specifying the `click` event type, only handlers for that event type will be unbound. This approach can still have negative ramifications if other scripts might be attaching behaviors to the same element, however. Robust and extensible applications typically demand the two-argument version for this reason:

```
1 | var handler = function() {  
2 |     alert('The quick brown fox jumps over the la  
3 | };  
4 | $('#foo').bind('click', handler);  
5 | $('#foo').unbind('click', handler);
```

By naming the handler, we can be assured that no other functions are accidentally removed. Note that the following will *not* work:

```
1 | $('#foo').bind('click', function() {  
2 |     alert('The quick brown fox jumps over the la  
3 | });  
4 |  
5 | // will NOT work  
6 | $('#foo').unbind('click', function() {  
7 |     alert('The quick brown fox jumps over the la  
8 | });
```



Even though the two functions are identical in content, they are created separately and so JavaScript is free to keep them as distinct function objects. To unbind a particular handler, we need a reference to that function and not a different one that happens to do the same thing.

**Note:** *Using a proxied function to unbind an event on an element will unbind all proxied functions on that element, as the same proxy function is used for all proxied events. To allow unbinding a specific event, use unique class names on the event (e.g. `click.proxy1`, `click.proxy2`) when attaching them.*

## Using Namespaces

Instead of maintaining references to handlers in order to unbind them, we can namespace the events and use this capability to narrow the scope of our unbinding actions. As shown in the discussion for the `.bind()` method, namespaces are defined by using a period (`.`) character when binding a handler:

```
1 | $('#foo').bind('click.myEvents', handler);
```

When a handler is bound in this fashion, we can still unbind it the normal way:

```
1 | $('#foo').unbind('click');
```

However, if we want to avoid affecting other handlers, we can be more specific:

```
1 | $('#foo').unbind('click.myEvents');
```

We can also unbind all of the handlers in a namespace, regardless of event type:

```
1 | $('#foo').unbind('.myEvents');
```

It is particularly useful to attach namespaces to event bindings when we are developing plug-ins or otherwise writing code that may interact with other event-handling code in the future.

### Using the Event Object

The third form of the `.unbind()` method is used when we wish to unbind a handler from within itself. For example, suppose we wish to trigger an event handler only three times:

```
1 | var timesClicked = 0;
2 | $('#foo').bind('click', function(event) {
3 |     alert('The quick brown fox jumps over the lazy dog');
4 |     timesClicked++;
5 |     if (timesClicked >= 3) {
6 |         $(this).unbind(event);
7 |     }
8 | });
```

The handler in this case must take a parameter, so that we can capture the event object and use it to unbind the handler after the third click. The event object contains the context necessary for `.unbind()` to know which handler to remove. This example is also an illustration of a closure. Since the handler refers to the `timesClicked`

variable, which is defined outside the function, incrementing the variable has an effect even between invocations of the handler.

## Examples:

**Example:** *Can bind and unbind events to the colored button.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5   button { margin:5px; }
6   button#theone { color:red; background:yellow;
7 </style>
8   <script src="http://code.jquery.com/jquery-
9 </head>
10 <body>
11   <button id="theone">Does nothing...</button>
12 <button id="bind">Bind Click</button>
13 <button id="unbind">Unbind Click</button>
14
15 <div style="display:none;">Click!</div>
16 <script>
17 function aClick() {
18   $("div").show().fadeOut("slow");
19 }
20 $("#bind").click(function () {
21   // could use .bind('click', aClick) instead b
22   $("#theone").click(aClick)
23   .text("Can Click!");
24 });
25 $("#unbind").click(function () {
26   $("#theone").unbind('click', aClick)
27   .text("Does nothing...");
28 });
29
30 </script>
```

```
31 |
32 | </body>
33 | </html>
```

## Demo

**Example:** *To unbind all events from all paragraphs, write:*

```
1 | $("p").unbind()
```

**Example:** *To unbind all click events from all paragraphs, write:*

```
1 | $("p").unbind( "click" )
```

**Example:** *To unbind just one previously bound handler, pass the function in as the second argument:*

```
1 | var foo = function () {
2 | // code to handle some kind of event
3 | };
4 |
5 | $("p").bind("click", foo); // ... now foo will
6 |
7 | $("p").unbind("click", foo); // ... foo will no
```

**A new version of this book is available!**

---

# .undelegate()

Categories: [Events](#) > [Event Handler Attachment](#)

`.undelegate()`

Returns: [jQuery](#)

**Description:** *Remove a handler from the event for all elements which match the current selector, based upon a specific set of root elements.*

**.undelegate()**

version added: [1.4.2](#)

This method does not accept any arguments.

**.undelegate( selector, eventType )**

version added: [1.4.2](#)

**selector**

Type: [String](#)

A selector which will be used to filter the event results.

**eventType**

Type: [String](#)

A string containing a JavaScript event type, such as "click" or "keydown"

**.undelegate( selector, eventType, handler(eventObject) )**

version added: [1.4.2](#)

**selector**

Type: [String](#)

A selector which will be used to filter the event results.

**eventType**

Type: [String](#)

A string containing a JavaScript event type, such as "click" or "keydown"

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute at the time the event is triggered.

**.undelegate( selector, events )**

version added: [1.4.3](#)

**selector**

Type: [String](#)

A selector which will be used to filter the event results.

### events

Type: [PlainObject](#)

An object of one or more event types and previously bound functions to unbind from them.

**`.undelegate( namespace )`**

version added: [1.6](#)

### namespace

Type: [String](#)

A string containing a namespace to unbind all events from.

The `.undelegate()` method is a way of removing event handlers that have been bound using `.delegate()`. **As of jQuery 1.7**, the `.on()` and `.off()` methods are preferred for attaching and removing event handlers.

## Examples:

**Example:** *Can bind and unbind events to the colored button.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5   button { margin:5px; }
6   button#theone { color:red; background:yellow;
7 </style>
8   <script src="http://code.jquery.com/jquery-
9 </head>
10 <body>
11   <button id="theone">Does nothing...</button>
12 <button id="bind">Bind Click</button>
13 <button id="unbind">Unbind Click</button>
14 <div style="display:none;">Click!</div>
15 <script>
16 function aClick() {
17   $("div").show().fadeOut("slow");
18 }
19 $("#bind").click(function () {
20   $("body").delegate("#theone", "click", aClick
21     .find("#theone").text("Can Click!");
22 });
23 $("#unbind").click(function () {
24   $("body").undelegate("#theone", "click", aC
25     .find("#theone").text("Does nothing...");
26 });
27 </script>
28
29 </body>
30 </html>
```

## Demo

**Example:** To unbind all delegated events from all paragraphs, write:

```
1 | $("p").undelegate()
```

**Example:** To unbind all delegated click events from all paragraphs, write:

```
1 | $("p").undelegate( "click" )
```

**Example:** To undelegate just one previously bound handler, pass the function in as the third argument:

```
1 | var foo = function () {  
2 |     // code to handle some kind of event  
3 | };  
4 |  
5 | // ... now foo will be called when paragraphs  
6 | $("body").delegate("p", "click", foo);  
7 |  
8 |  
9 | // ... foo will no longer be called.  
10 | $("body").undelegate("p", "click", foo);
```

**Example:** To unbind all delegated events by their namespace:

```
1 | var foo = function () {  
2 |     // code to handle some kind of event
```

```
3 };
4
5 // delegate events under the ".whatever" namespace
6 $("form").delegate(":button", "click.whatever", foo);
7
8 $("form").delegate("input[type='text']", "keypress.whatever", foo);
9
10 // unbind all events delegated under the ".whatever" namespace
11
12 $("form").undelegate(".whatever");
```

POWERED BYHERONOTE

A new version of this book is available!

---

# .unload()

Categories: [Events](#) > [Document Loading](#)

`.unload( handler(eventObject) )`

Returns: [jQuery](#)  
version deprecated: [1.8](#)

**Description:** *Bind an event handler to the "unload" JavaScript event.*

`.unload( handler(eventObject) )`

version added: [1.0](#)

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute when the event is triggered.

`.unload( [ eventData ],  
handler(eventObject) )`

version added: [1.4.3](#)

**eventData**

Type: [Object](#)

A plain object of data that will be passed to the event handler.

**handler(eventObject)**

Type: [Function\(\)](#)

A function to execute each time the event is triggered.

This method is a shortcut for `.on('unload', handler)`.

The `unload` event is sent to the `window` element when the user navigates away from the page. This could mean one of many things. The user could have clicked on a link to leave the page, or typed in a new URL in the address bar. The forward and back buttons will trigger the event. Closing the browser window will cause the event to be triggered. Even a page reload will first create an `unload` event.

*The exact handling of the `unload` event has varied from version to version of browsers. For example, some versions of Firefox trigger the event when a link is followed, but*

*not when the window is closed. In practical usage, behavior should be tested on all supported browsers, and contrasted with the proprietary `beforeunload` event.*

Any `unload` event handler should be bound to the `window` object:

```
1 | $(window).unload(function() {  
2 |     alert('Handler for .unload() called.');
```

```
3 | });
```

After this code executes, the alert will be displayed whenever the browser leaves the current page. It is not possible to cancel the `unload` event with `.preventDefault()`. This event is available so that scripts can perform cleanup when the user leaves the page.

## Example:

*To display an alert when a page is unloaded:*

```
1 | $(window).unload( function () { alert("Bye now
```

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# .unwrap()

Categories: [Manipulation](#) > [DOM Insertion, Around](#) | [Manipulation](#) > [DOM Removal](#)

`.unwrap()`

Returns: [jQuery](#)

**Description:** *Remove the parents of the set of matched elements from the DOM, leaving the matched elements in their place.*

`.unwrap()`

version added: [1.4](#)

This method does not accept any arguments.

The `.unwrap()` method removes the element's parent. This is effectively the inverse of the `.wrap()` method. The matched elements (and their siblings, if any) replace their parents within the DOM structure.

## Example:

*Wrap/unwrap a div around each of the paragraphs.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5    div { border: 2px solid blue; }
6    p { background:yellow; margin:4px; }
7    </style>
8    <script src="http://code.jquery.com/jquery-
9  </head>
10 <body>
11   <button>wrap/unwrap</button>
12   <p>Hello</p>
13   <p>cruel</p>
14   <p>World</p>
15   <script>
16   var pTags = $("p");
17   $("button").click( function(){
18     if ( pTags.parent().is("div") ) {
19       pTags.unwrap();
20     } else {
21       pTags.wrap("<div></div>");
22     }
23   });</script>
24
25 </body>
26 </html>
```

Demo

A new version of this book is available!

---

# .val()

Categories: [Attributes](#) | [Forms](#) | [Manipulation](#) > [General Attributes](#)

Get the current value of the first element in the set of matched elements or set the value of every matched element.

## Contents:

[.val\(\)](#)

`.val()`

[.val\( value \)](#)

`.val( value )`

`.val( function(index, value) )`

---

`.val()`

Returns: [String](#), [Number](#), [Array](#)

**Description:** *Get the current value of the first element in the set of matched elements.*

`.val()`

version added: [1.0](#)

This method does not accept any arguments.

The `.val()` method is primarily used to get the values of form elements such as `input`, `select` and `textarea`. In the case of `<select multiple="multiple">` elements, the `.val()` method returns an array containing each selected option; if no option is selected, it returns `null`.

For selects and checkboxes, you can also use the `:selected` and `:checked` selectors to get at values, for example:

```
1 | $( 'select.foo option:selected' ).val(); // (
2 | $( 'select.foo' ).val(); // (
3 | $( 'input:checkbox:checked' ).val(); // (
4 | $( 'input:radio[name=bar]:checked' ).val(); // (
```

**Note:** *At present, using `.val()` on `textarea` elements strips carriage return characters from the browser-reported value. When this value is sent to the server via XHR however, carriage returns are preserved (or added by browsers which do not include them in the raw value). A workaround for this issue can be achieved using a `valHook`*

*as follows:*

```
1 | $.valHooks.textarea = {  
2 |   get: function( elem ) {  
3 |     return elem.value.replace( /\r?\n/g, "\r\n"  
4 |   }  
5 | };
```

## Examples:

**Example:** Get the single value from a single select and an array of values from a multiple select and display their values.

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5      p { color:red; margin:4px; }
6      b { color:blue; }
7    </style>
8    <script src="http://code.jquery.com/jquery-
9  </head>
10 <body>
11   <p></p>
12
13   <select id="single">
14     <option>Single</option>
15     <option>Single2</option>
16   </select>
17
18   <select id="multiple" multiple="multiple">
19     <option selected="selected">Multiple</opt
20     <option>Multiple2</option>
21     <option selected="selected">Multiple3</op
22   </select>
23
24 <script>
25   function displayVals() {
26     var singleValues = $("#single").val();
27     var multipleValues = $("#multiple").val
28     $("p").html("<b>Single:</b> " +
29               singleValues +
```

```
30         " <b>Multiple:</b> " +
31         multipleValues.join(", "));
32     }
33
34     $("select").change(displayVals);
35     displayVals();
36
37 </script>
38
39 </body>
40 </html>
```

Demo

**Example:** *Find the value of an input box.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5
6   p { color:blue; margin:8px; }
7 </style>
8   <script src="http://code.jquery.com/jquery-
9 </head>
10 <body>
11   <input type="text" value="some text"/>
12   <p></p>
13 <script>
14   $("input").keyup(function () {
15     var value = $(this).val();
16     $("p").text(value);
17   }).keyup();
18 </script>
19
20 </body>
```

```
21 | </html>
```

Demo

`.val( value )`

Returns: [jQuery](#)

**Description:** *Set the value of each element in the set of matched elements.*

`.val( value )`

version added: [1.0](#)

**value**

Type: [String](#) or [Array](#)

A string of text or an array of strings corresponding to the value of each matched element to set as selected/checked.

`.val( function(index, value) )`

version added: [1.4](#)

**function(index, value)**

Type: [Function\(\)](#)

A function returning the value to set. `this` is the current element. Receives the index position of the element in the set and the old value as arguments.

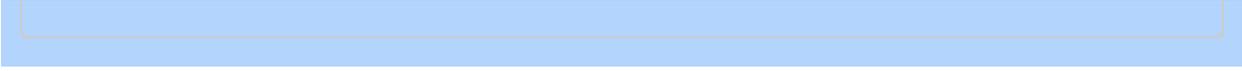
This method is typically used to set the values of form fields.

Passing an array of element values allows matching `<input type="checkbox">`, `<input type="radio">` and `<option>`s inside of n `<select multiple="multiple">` to be selected. In the case of `<input type="radio">`s that are part of a radio group and `<select multiple="multiple">` the other elements will be deselected.

The `.val()` method allows us to set the value by passing in a function. As of jQuery 1.4, the function is passed two arguments, the current element's index and its current value:

```
1 | $( 'input:text.items' ).val( function( index, va
2 |     return value + ' ' + this.className;
3 | });
```

This example appends the string " items" to the text inputs' values.



## Examples:

**Example:** *Set the value of an input box.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5      button { margin:4px; cursor:pointer; }
6      input { margin:4px; color:blue; }
7    </style>
8    <script src="http://code.jquery.com/jquery-
9  </head>
10 <body>
11   <div>
12     <button>Feed</button>
13     <button>the</button>
14     <button>Input</button>
15   </div>
16   <input type="text" value="click a button" />
17 <script>
18   $("button").click(function () {
19     var text = $(this).text();
20     $("input").val(text);
21   });
22 </script>
23
24 </body>
25 </html>
```

Demo

**Example:** *Use the function argument to modify the value of an input box.*

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://code.jquery.com/jquery-latest.js">
5 </head>
6 <body>
7
8   <p>Type something and then click or tab out of the inp
9   <input type="text" value="type something" />
10
11 <script>
12   $('input').on('blur', function() {
13     $(this).val(function( i, val ) {
14       return val.toUpperCase();
15     });
16   });
17 </script>
18
19 </body>
20 </html>

```

Demo

**Example:** Set a single select, a multiple select, checkboxes and a radio button .

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5     body { color:blue; }
6   </style>
7   <script src="http://code.jquery.com/jquery-
8 </head>
9 <body>

```

```
10 <select id="single">
11   <option>Single</option>
12   <option>Single2</option>
13 </select>
14
15 <select id="multiple" multiple="multiple">
16   <option selected="selected">Multiple</option>
17   <option>Multiple2</option>
18   <option selected="selected">Multiple3</option>
19 </select><br/>
20 <input type="checkbox" name="checkboxname" value
21 <input type="checkbox" name="checkboxname" value
22 <input type="radio" name="r" value="radio1"/>
23 <input type="radio" name="r" value="radio2"/>
24 <script>
25
26   $("#single").val("Single2");
27   $("#multiple").val(["Multiple2", "Multiple3"]
28   $("input").val(["check1", "check2", "radio1"]
29
30 </script>
31
32 </body>
33 </html>
```

Demo

POWERED BY HERONOTE

A new version of this book is available!

---

# :visible Selector

Categories: [Selectors](#) > [jQuery Extensions](#) | [Selectors](#) > [Visibility Filter](#)

## visible selector

**Description:** *Selects all elements that are visible.*

**jQuery( ":visible" )**

version added: [1.0](#)

Elements are considered visible if they consume space in the document. Visible elements have a width or height that is greater than zero.

Elements with `visibility: hidden` or `opacity: 0` are considered visible, since they still consume space in the layout. During animations that hide an element, the element is considered to be visible until the end of the animation. During animations to show an element, the element is considered to be visible at the start at the animation.

How `:visible` is calculated was changed in jQuery 1.3.2. The [release notes](#) outline the changes in more detail.

### Additional Notes:

Because `:visible` is a jQuery extension and not part of the CSS specification, queries using `:visible` cannot take advantage of the performance boost provided by the native DOM `querySelectorAll()` method. To achieve the best performance when using `:visible` to select elements, first select the elements using a pure CSS selector, then use `.filter(":visible")`.

## Example:

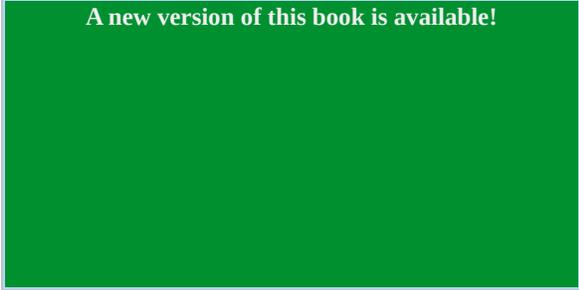
*Make all visible divs turn yellow on click.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5    div { width:50px; height:40px; margin:5px; }
6    .starthidden { display:none; }
7  </style>
8  <script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
9  </head>
10 <body>
11   <button>Show hidden to see they don't change
12   <div></div>
13   <div class="starthidden"></div>
14   <div></div>
15
16   <div></div>
17   <div style="display:none;"></div>
18 <script>
19   $("div:visible").click(function () {
20     $(this).css("background", "yellow");
21   });
22   $("button").click(function () {
23     $("div:hidden").show("fast");
24   });
25
26 </script>
27
28 </body>
29 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

# .width()

Categories: [CSS](#) | [Dimensions](#) | [Manipulation](#) > [Style Properties](#)

Get the current computed width for the first element in the set of matched elements or set the width of every matched element.

## Contents:

[.width\(\)](#)

`.width()`

[.width\( value \)](#)

`.width( value )`

`.width( function(index, width) )`

---

`.width()`

Returns: `Integer`

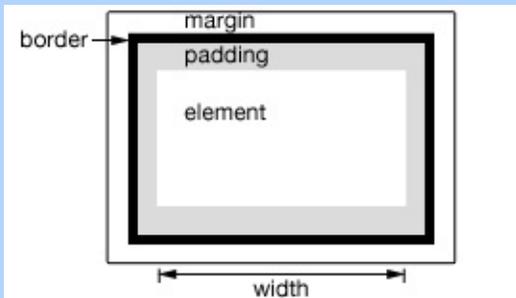
**Description:** *Get the current computed width for the first element in the set of matched elements.*

`.width()`

version added: `1.0`

This method does not accept any arguments.

The difference between `.css(width)` and `.width()` is that the latter returns a unit-less pixel value (for example, `400`) while the former returns a value with units intact (for example, `400px`). The `.width()` method is recommended when an element's width needs to be used in a mathematical calculation.



This method is also able to find the width of the window and document.

```
1 | $(window).width(); // returns width of browser window  
2 | $(document).width(); // returns width of document
```

Note that `.width()` will always return the content width, regardless of the value of the CSS `box-sizing` property.



```
30     showWidth("document", $(document).width());
31 });
32 $("#getw").click(function () {
33     showWidth("window", $(window).width());
34 });
35
36 </script>
37
38 </body>
39 </html>
```

Demo

`.width( value )`

Returns: [jQuery](#)

**Description:** *Set the CSS width of each element in the set of matched elements.*

`.width( value )`

version added: [1.0](#)

**value**

Type: [String](#) or [Number](#)

An integer representing the number of pixels, or an integer along with an optional unit of measure appended (as a string).

`.width( function(index, width) )`

version added: [1.4.1](#)

**function(index, width)**

Type: [Function\(\)](#)

A function returning the width to set. Receives the index position of the element in the set and the old width as arguments. Within the function, `this` refers to the current element in the set.

When calling `.width("value")`, the value can be either a string (number and unit) or a number. If only a number is provided for the value, jQuery assumes a pixel unit. If a string is provided, however, any valid CSS measurement may be used for the width (such as `100px`, `50%`, or `auto`). Note that in modern browsers, the CSS width property does not include padding, border, or margin, unless the `box-sizing` CSS property is used.

If no explicit unit is specified (like "em" or "%") then "px" is assumed.

Note that `.width("value")` sets the content width of the box regardless of the value of the CSS `box-sizing` property.

## Example:

*Change the width of each div the first time it is clicked (and change its color).*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5   div { width: 70px; height: 50px; float:left;
6         background: red; cursor: pointer; }
7   .mod { background: blue; cursor: default; }
8   </style>
9   <script src="http://code.jquery.com/jquery-
10 </head>
11 <body>
12
13   <div>d</div>
14   <div>d</div>
15   <div>d</div>
16   <div>d</div>
17   <div>d</div>
18
19 <script>
20 (function() {
21   var modWidth = 50;
22   $("div").one('click', function () {
23     $(this).width(modWidth).addClass("mod");
24     modWidth -= 8;
25   });
26 })();
27 </script>
28
29 </body>
30 </html>
```

Demo

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# .wrap()

Categories: [Manipulation](#) > [DOM Insertion, Around](#)

`.wrap( wrappingElement )`

Returns: [jQuery](#)

**Description:** *Wrap an HTML structure around each element in the set of matched elements.*

`.wrap( wrappingElement )`

version added: [1.0](#)

**wrappingElement**

Type: [Selector](#) or [htmlString](#) or [Element](#) or [jQuery](#)

A selector, element, HTML string, or jQuery object specifying the structure to wrap around the matched elements.

`.wrap( function(index) )`

version added: [1.4](#)

**function(index)**

Type: [Function\(\)](#)

A callback function returning the HTML content or jQuery object to wrap around the matched elements. Receives the index position of the element in the set as an argument. Within the function, `this` refers to the current element in the set.

The `.wrap()` function can take any string or object that could be passed to the `$( )` factory function to specify a DOM structure. This structure may be nested several levels deep, but should contain only one inmost element. A copy of this structure will be wrapped around each of the elements in the set of matched elements. This method returns the original set of elements for chaining purposes.

Consider the following HTML:

```
1 | <div class="container">
2 |   <div class="inner">Hello</div>
3 |   <div class="inner">Goodbye</div>
4 | </div>
```

Using `.wrap()`, we can insert an HTML structure around the inner `<div>` elements like so:

```
1 | $(' .inner ').wrap( '<div class="new" />' );
```

The new `<div>` element is created on the fly and added to the DOM. The result is a new `<div>` wrapped around each matched element:

```
1 | <div class="container">
2 |   <div class="new">
3 |     <div class="inner">Hello</div>
4 |   </div>
5 |   <div class="new">
6 |     <div class="inner">Goodbye</div>
7 |   </div>
8 | </div>
```

The second version of this method allows us to instead specify a callback function. This callback function will be called once for every matched element; it should return a DOM element, jQuery object, or HTML snippet in which to wrap the corresponding element. For example:

```
1 | $(' .inner ').wrap(function() {
2 |   return '<div class="' + $(this).text() + '"
3 | });
```

This will cause each `<div>` to have a class corresponding to the text it wraps:

```
1 | <div class="container">
2 |   <div class="Hello">
3 |     <div class="inner">Hello</div>
4 |   </div>
5 |   <div class="Goodbye">
```

```
6     <div class="inner">Goodbye</div>
7     </div>
8 </div>
```

## Examples:

**Example:** *Wrap a new div around all of the paragraphs.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5
6    div { border: 2px solid blue; }
7    p { background:yellow; margin:4px; }
8  </style>
9  <script src="http://code.jquery.com/jquery-
10 </head>
11 <body>
12   <p>Hello</p>
13   <p>cruel</p>
14   <p>World</p>
15 <script>$("p").wrap("<div></div>");</script>
16
17 </body>
18 </html>
```

### Demo

**Example:** *Wraps a newly created tree of objects around the spans. Notice anything in between the spans gets left out like the <strong> (red text) in this example. Even the white space between spans is left out. Click View Source to see the original html.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
```

```

4   <style>
5
6   div { border:2px blue solid; margin:2px; padding
7   p { background:yellow; margin:2px; padding:2px;
8   strong { color:red; }
9   </style>
10  <script src="http://code.jquery.com/jquery-latest.js"
11 </head>
12 <body>
13   <span>Span Text</span>
14   <strong>What about me?</strong>
15   <span>Another One</span>
16   <script>$("span").wrap("<div><div><p><em><b></b></em></p>
17
18 </body>
19 </html>

```

Demo

**Example:** *Wrap a new div around all of the paragraphs.*

```

1   <!DOCTYPE html>
2   <html>
3   <head>
4     <style>
5
6     div { border: 2px solid blue; }
7     p { background:yellow; margin:4px; }
8   </style>
9   <script src="http://code.jquery.com/jquery-
10 </head>
11 <body>
12   <p>Hello</p>
13   <p>cruel</p>
14   <p>World</p>
15   <script>$("p").wrap(document.createElement("d.

```

```
16  
17 </body>  
18 </html>
```

## Demo

**Example:** *Wrap a jQuery object double depth div around all of the paragraphs. Notice it doesn't move the object but just clones it to wrap around its target.*

```
1 <!DOCTYPE html>  
2 <html>  
3 <head>  
4 <style>  
5  
6 div { border: 2px solid blue; margin:2px; padding: 2px; }  
7 .doublediv { border-color:red; }  
8 p { background:yellow; margin:4px; font-size: 1.2em; }  
9 </style>  
10 <script src="http://code.jquery.com/jquery-1.10.2.min.js"></script>  
11 </head>  
12 <body>  
13 <p>Hello</p>  
14 <p>cruel</p>  
15 <p>world</p>  
16 <div class="doublediv"><div></div></div>  
17 <script>$("p").wrap($(".doublediv"));</script>  
18  
19 </body>  
20 </html>
```

## Demo

**A new version of this book is available!**

---

# .wrapAll()

Categories: [Manipulation](#) > [DOM Insertion, Around](#)

`.wrapAll( wrappingElement )`

Returns: `jQuery`

**Description:** *Wrap an HTML structure around all elements in the set of matched elements.*

`.wrapAll( wrappingElement )`

version added: `1.2`

### **wrappingElement**

Type: `Selector` or `htmlString` or `Element` or `jQuery`

A selector, element, HTML string, or jQuery object specifying the structure to wrap around the matched elements.

The `.wrapAll()` function can take any string or object that could be passed to the `$( )` function to specify a DOM structure. This structure may be nested several levels deep, but should contain only one inmost element. The structure will be wrapped around all of the elements in the set of matched elements, as a single group.

Consider the following HTML:

```
1 | <div class="container">
2 |   <div class="inner">Hello</div>
3 |   <div class="inner">Goodbye</div>
4 | </div>
```

Using `.wrapAll()`, we can insert an HTML structure around the inner `<div>` elements like so:

```
1 | $(' .inner ').wrapAll( '<div class="new" />' );
```

The new `<div>` element is created on the fly and added to the DOM. The result is a new `<div>` wrapped around all matched elements:

```
1 | <div class="container">
```

```
2 <div class="new">
3   <div class="inner">Hello</div>
4   <div class="inner">Goodbye</div>
5 </div>
6 </div>
```

## Examples:

**Example:** *Wrap a new div around all of the paragraphs.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5
6    div { border: 2px solid blue; }
7    p { background:yellow; margin:4px; }
8  </style>
9  <script src="http://code.jquery.com/jquery-
10 </head>
11 <body>
12   <p>Hello</p>
13   <p>cruel</p>
14   <p>World</p>
15 <script>$("p").wrapAll("<div></div>");</scrip
16
17 </body>
18 </html>
```

### Demo

**Example:** *Wraps a newly created tree of objects around the spans. Notice anything in between the spans gets left out like the <strong> (red text) in this example. Even the white space between spans is left out. Click View Source to see the original html.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
```

```

4   <style>
5
6   div { border:2px blue solid; margin:2px; padding
7   p { background:yellow; margin:2px; padding:2px;
8   strong { color:red; }
9   </style>
10  <script src="http://code.jquery.com/jquery-latest.js"
11 </head>
12 <body>
13   <span>Span Text</span>
14   <strong>What about me?</strong>
15   <span>Another One</span>
16  <script>$("span").wrapAll("<div><div><p><em><b></b></em>
17
18 </body>
19 </html>

```

Demo

**Example:** *Wrap a new div around all of the paragraphs.*

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>
5
6    div { border: 2px solid blue; }
7    p { background:yellow; margin:4px; }
8  </style>
9  <script src="http://code.jquery.com/jquery-
10 </head>
11 <body>
12   <p>Hello</p>
13   <p>cruel</p>
14   <p>World</p>
15  <script>$("p").wrapAll(document.createElement

```

```
16  
17 </body>  
18 </html>
```

## Demo

**Example:** *Wrap a jQuery object double depth div around all of the paragraphs. Notice it doesn't move the object but just clones it to wrap around its target.*

```
1 <!DOCTYPE html>  
2 <html>  
3 <head>  
4 <style>  
5  
6 div { border: 2px solid blue; margin:2px; padding: 2px; }  
7 .doublediv { border-color:red; }  
8 p { background:yellow; margin:4px; font-size: 1.2em; }  
9 </style>  
10 <script src="http://code.jquery.com/jquery-1.10.2.min.js"></script>  
11 </head>  
12 <body>  
13 <p>Hello</p>  
14 <p>cruel</p>  
15 <p>world</p>  
16 <div class="doublediv"><div></div></div>  
17 <script>$("p").wrapAll($(".doublediv"));</script>  
18  
19 </body>  
20 </html>
```

## Demo

**A new version of this book is available!**

---

# .wrapInner()

Categories: [Manipulation](#) > [DOM Insertion, Around](#)

`.wrapInner( wrappingElement )`

Returns: [jQuery](#)

**Description:** *Wrap an HTML structure around the content of each element in the set of matched elements.*

`.wrapInner( wrappingElement )`

version added: [1.2](#)

**wrappingElement**

Type: [String](#)

An HTML snippet, selector expression, jQuery object, or DOM element specifying the structure to wrap around the content of the matched elements.

`.wrapInner( function(index) )`

version added: [1.4](#)

**function(index)**

Type: [Function\(\)](#)

A callback function which generates a structure to wrap around the content of the matched elements. Receives the index position of the element in the set as an argument. Within the function, `this` refers to the current element in the set.

The `.wrapInner()` function can take any string or object that could be passed to the `$( )` factory function to specify a DOM structure. This structure may be nested several levels deep, but should contain only one inmost element. The structure will be wrapped around the content of each of the elements in the set of matched elements.

Consider the following HTML:

```
1 | <div class="container">
2 |   <div class="inner">Hello</div>
3 |   <div class="inner">Goodbye</div>
4 | </div>
```

Using `.wrapInner()`, we can insert an HTML structure around the content of each inner `<div>` elements like so:

```
1 | $(' .inner ' ).wrapInner( '<div class="new" />' );
```

The new `<div>` element is created on the fly and added to the DOM. The result is a new `<div>` wrapped around the content of each matched element:

```
1 | <div class="container">
2 |   <div class="inner">
3 |     <div class="new">Hello</div>
4 |   </div>
5 |   <div class="inner">
6 |     <div class="new">Goodbye</div>
7 |   </div>
8 | </div>
```

The second version of this method allows us to instead specify a callback function. This callback function will be called once for every matched element; it should return a DOM element, jQuery object, or HTML snippet in which to wrap the content of the corresponding element. For example:

```
1 | $(' .inner ' ).wrapInner( function() {
2 |   return '<div class="' + this.nodeValue + "'
3 | });
```

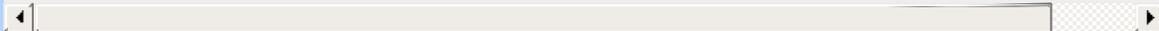
This will cause each `<div>` to have a class corresponding to the text it wraps:

```
1 | <div class="container">
2 |   <div class="inner">
3 |     <div class="Hello">Hello</div>
4 |   </div>
```

```
5 | <div class="inner">
6 |   <div class="Goodbye">Goodbye</div>
7 | </div>
8 | </div>
```

**Note:** When passing a selector string to the `.wrapInner()` function, the expected input is well formed HTML with correctly closed tags. Examples of valid input include:

```
1 | $(elem).wrapInner("<div class='test' />");
2 | $(elem).wrapInner("<div class='test'></div>");
3 | $(elem).wrapInner("<div class=\"test\"></div>");
```



## Examples:

**Example:** *Selects all paragraphs and wraps a bold tag around each of its contents.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>p { background:#bbf; }</style>
5   <script src="http://code.jquery.com/jquery-
6 </head>
7 <body>
8   <p>Hello</p>
9
10  <p>cruel</p>
11  <p>World</p>
12  <script>$("p").wrapInner("<b></b>");</script>
13
14 </body>
15 </html>
```

### Demo

**Example:** *Wraps a newly created tree of objects around the inside of the body.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5
6   div { border:2px green solid; margin:2px; p
7   p { background:yellow; margin:2px; padding:-
8   </style>
```

```
9   <script src="http://code.jquery.com/jquery-latest.js">
10  </head>
11  <body>
12    Plain old text, or is it?
13    <script>$("body").wrapInner("<div><div><p><em><b></b></em></p>");
14
15  </body>
16  </html>
```

### Demo

**Example:** *Selects all paragraphs and wraps a bold tag around each of its contents.*

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <style>p { background:#9f9; }</style>
5    <script src="http://code.jquery.com/jquery-
6  </head>
7  <body>
8    <p>Hello</p>
9
10   <p>cruel</p>
11   <p>world</p>
12   <script>$("p").wrapInner(document.createElement("b"));
13
14  </body>
15  </html>
```

### Demo

**Example:** *Selects all paragraphs and wraps a jQuery object around each of its contents.*

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <style>
5
6   p { background:#9f9; }
7   .red { color:red; }
8 </style>
9   <script src="http://code.jquery.com/jquery-latest.js">
10 </script>
11 </head>
12 <body>
13   <p>Hello</p>
14   <p>cruel</p>
15   <p>world</p>
16   <script>$("p").wrapInner("<span class='red'></span>")
17 </script>
18 </body>
</html>
```

Demo

POWERED BY HERONOTE

A new version of this book is available!

---

# Category: Ajax

The jQuery library has a full suite of AJAX capabilities. The functions and methods therein allow us to load data from the server without a browser page refresh.

Also in: [Ajax](#) > [Global Ajax Event Handlers](#)

---

## **.ajaxComplete()**

Register a handler to be called when Ajax requests complete. This is an AjaxEvent.

Also in: [Ajax > Global Ajax Event Handlers](#)

---

## **.ajaxError()**

Register a handler to be called when Ajax requests complete with an error. This is an Ajax Event.

Also in: [Ajax > Global Ajax Event Handlers](#)

---

## **.ajaxSend()**

Attach a function to be executed before an Ajax request is sent.  
This is an Ajax Event.

Also in: [Ajax > Global Ajax Event Handlers](#)

---

## **.ajaxStart()**

Register a handler to be called when the first Ajax request begins. This is an Ajax Event.

Also in: [Ajax > Global Ajax Event Handlers](#)

---

## **.ajaxStop()**

Register a handler to be called when all Ajax requests have completed. This is an Ajax Event.

Also in: [Ajax > Global Ajax Event Handlers](#)

---

## **.ajaxSuccess()**

Attach a function to be executed whenever an Ajax request completes successfully. This is an Ajax Event.

Also in: [Ajax > Low-Level Interface](#)

---

## jQuery.ajax()

Perform an asynchronous HTTP (Ajax) request.

Also in: [Ajax](#) > [Low-Level Interface](#)

---

## jQuery.ajaxPrefilter()

Handle custom Ajax options or modify existing options before each request is sent and before they are processed by \$.ajax().

Also in: [Ajax > Low-Level Interface](#)

---

## jQuery.ajaxSetup()

Set default values for future Ajax requests.

Also in: [Ajax](#) > [Low-Level Interface](#)

---

## jQuery.ajaxTransport()

Creates an object that handles the actual transmission of Ajax data.

Also in: [Ajax > Shorthand Methods](#)

---

## jQuery.get()

Load data from the server using a HTTP GET request.

Also in: [Ajax](#) > [Shorthand Methods](#)

---

## jQuery.getJSON()

Load JSON-encoded data from the server using a GET HTTP request.

Also in: [Ajax](#) > [Shorthand Methods](#)

---

## jQuery.getScript()

Load a JavaScript file from the server using a GET HTTP request, then execute it.

Also in: [Miscellaneous](#) > [Collection Manipulation](#) | [Forms](#) | [Ajax](#) > [Helper Functions](#)

---

## jQuery.param()

Create a serialized representation of an array or object, suitable for use in a URL query string or Ajax request.

Also in: [Ajax > Shorthand Methods](#)

---

## jQuery.post()

Load data from the server using a HTTP POST request.

Also in: [Ajax](#) > [Shorthand Methods](#)

---

## **.load()**

Load data from the server and place the returned HTML into the matched element.

Also in: [Forms](#) | [Ajax > Helper Functions](#)

---

## **.serialize()**

Encode a set of form elements as a string for submission.

Also in: [Forms](#) | [Ajax > Helper Functions](#)

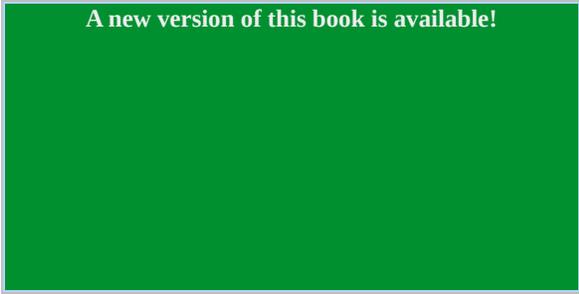
---

## **.serializeArray()**

Encode a set of form elements as an array of names and values.

POWERED BY **HERONOTE**

A new version of this book is available!



---

## Category: Global Ajax Event Handlers

These methods register handlers to be called when certain events, such as initialization or completion, take place for any AJAX request on the page. The global events are fired on each AJAX request if the `global` property in `jQuery.ajaxSetup()` is `true`, which it is by default.

*Note: Global events are never fired for cross-domain script or JSONP requests, regardless of the value of `global`.*

---

## **.ajaxComplete()**

Register a handler to be called when Ajax requests complete. This is an AjaxEvent.

---

## **.ajaxError()**

Register a handler to be called when Ajax requests complete with an error. This is an Ajax Event.

---

## **.ajaxSend()**

Attach a function to be executed before an Ajax request is sent.  
This is an Ajax Event.

---

## **.ajaxStart()**

Register a handler to be called when the first Ajax request begins.  
This is an Ajax Event.

---

## **.ajaxStop()**

Register a handler to be called when all Ajax requests have completed. This is an Ajax Event.

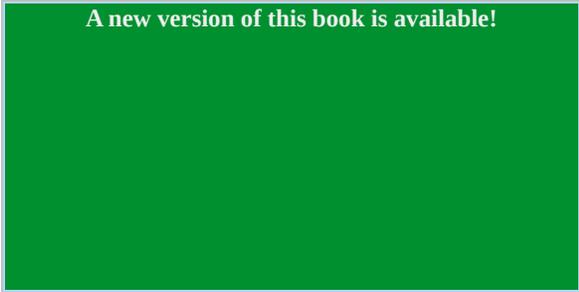
---

## **.ajaxSuccess()**

Attach a function to be executed whenever an Ajax request completes successfully. This is an Ajax Event.

POWERED BY **HERONOTE**

A new version of this book is available!



---

# Category: Helper Functions

These functions assist with common idioms encountered when performing AJAX tasks.

Also in: [Miscellaneous](#) > [Collection Manipulation](#) | [Forms](#)

---

## jQuery.param()

Create a serialized representation of an array or object, suitable for use in a URL query string or Ajax request.

Also in: [Forms](#)

---

## **.serialize()**

Encode a set of form elements as a string for submission.

Also in: [Forms](#)

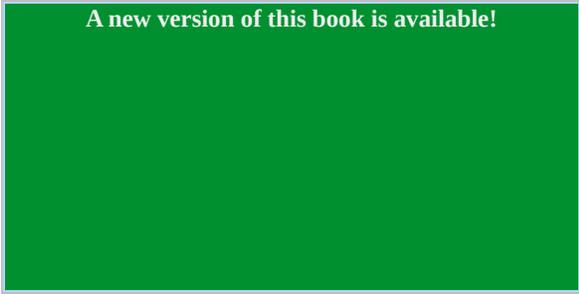
---

## **.serializeArray()**

Encode a set of form elements as an array of names and values.

POWERED BY **HERONOTE**

A new version of this book is available!



---

## Category: Low-Level Interface

These methods can be used to make arbitrary AJAX requests.

---

## jQuery.ajax()

Perform an asynchronous HTTP (Ajax) request.

---

## jQuery.ajaxPrefilter()

Handle custom Ajax options or modify existing options before each request is sent and before they are processed by \$.ajax().

---

## jQuery.ajaxSetup()

Set default values for future Ajax requests.

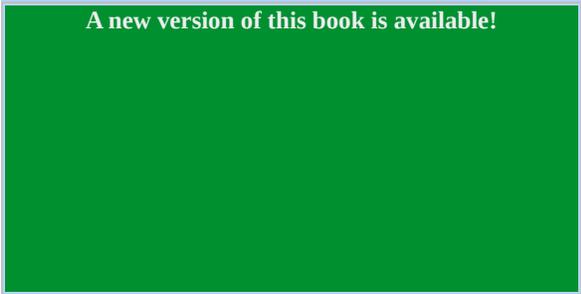
---

## jQuery.ajaxTransport()

Creates an object that handles the actual transmission of Ajax data.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

## Category: Shorthand Methods

These methods perform the more common types of AJAX requests in less code.

---

## jQuery.get()

Load data from the server using a HTTP GET request.

---

## jQuery.getJSON()

Load JSON-encoded data from the server using a GET HTTP request.

---

## jQuery.getScript()

Load a JavaScript file from the server using a GET HTTP request, then execute it.

---

## jQuery.post()

Load data from the server using a HTTP POST request.

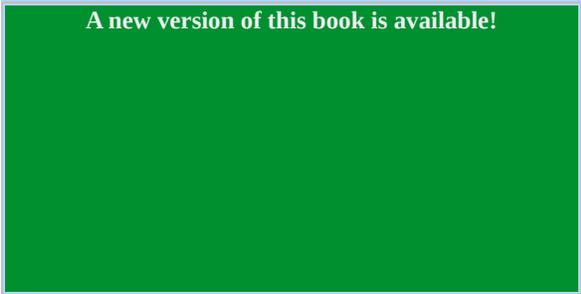
---

## **.load()**

Load data from the server and place the returned HTML into the matched element.

POWERED BY **HERONOTE**

A new version of this book is available!



# Ajax Events

Ajax requests produce a number of different events that you can subscribe to. Here's a full list of the events and in what order they are broadcast.

There are two types of events:

## Local Events

These are callbacks that you can subscribe to within the Ajax request object, like so:

```
1 | $.ajax({
2 |   beforeSend: function(){
3 |     // Handle the beforeSend event
4 |   },
5 |   complete: function(){
6 |     // Handle the complete event
7 |   }
8 |   // .....
9 | });
```

## Global Events

These events are broadcast to all elements in the DOM, triggering any handlers which may be listening. You can listen for these events like so:

```
$("#loading").bind("ajaxSend", function(){
  $(this).show();
}).bind("ajaxComplete", function(){
  $(this).hide();
});
```

Global events can be disabled for a particular Ajax request by passing in the global option, like so:

```
1 | $.ajax({
2 |   url: "test.html",
3 |   global: false,
4 |   // ...
5 | });
```

## Events

This is the full list of Ajax events that are broadcast, and in the order in which they are broadcast. The indented events are broadcast for each and every Ajax request (unless a global option has been set). The `ajaxStart` and `ajaxStop` events are events that relate to all Ajax requests together.

### **ajaxStart** (Global Event)

This event is broadcast if an Ajax request is started and no other Ajax requests are currently running.

### **beforeSend** (Local Event)

This event, which is triggered before an Ajax request is started, allows you to modify the XMLHttpRequest object (setting additional headers, if need be.)

### **ajaxSend** (Global Event)

This global event is also triggered before the request is run.

### **success** (Local Event)

This event is only called if the request was successful (no errors from the server, no errors with the data).

### **ajaxSuccess** (Global Event)

This event is also only called if the request was successful.

### **error** (Local Event)

This event is only called if an error occurred with the request (you can never have both an error and a success callback with a request).

### **ajaxError** (Global Event)

This global event behaves the same as the local error event.

### **complete** (Local Event)

This event is called regardless of if the request was successful, or not. You will always receive a complete

callback, even for synchronous requests.

**ajaxComplete** (Global Event)

This event behaves the same as the complete event and will be triggered every time an Ajax request finishes.

**ajaxStop** (Global Event)

This global event is triggered if there are no more Ajax requests being processed.

POWERED BY [HERONOTE](#)

A new version of this book is available!

---

# Category: Attributes

These methods get and set DOM attributes of elements.

Also in: [Manipulation](#) > [Class Attribute](#) | [CSS](#)

---

## **.addClass()**

Adds the specified class(es) to each of the set of matched elements.

Also in: [Manipulation](#) > [General Attributes](#)

---

## **.attr()**

Get the value of an attribute for the first element in the set of matched elements or set one or more attributes for every matched element.

Also in: [Manipulation](#) > [Class Attribute](#) | [CSS](#)

---

## **.hasClass()**

Determine whether any of the matched elements are assigned the given class.

Also in: [Manipulation](#) > [DOM Insertion, Inside](#)

---

## `.html()`

Get the HTML contents of the first element in the set of matched elements or set the HTML contents of every matched element.

Also in: [Manipulation](#) > [General Attributes](#)

---

## **.prop()**

Get the value of a property for the first element in the set of matched elements or set one or more properties for every matched element.

Also in: [Manipulation](#) > [General Attributes](#)

---

## **.removeAttr()**

Remove an attribute from each element in the set of matched elements.

Also in: [Manipulation > Class Attribute](#) | [CSS](#)

---

## **.removeClass()**

Remove a single class, multiple classes, or all classes from each element in the set of matched elements.

Also in: [Manipulation](#) > [General Attributes](#)

---

## `.removeProp()`

Remove a property for the set of matched elements.

Also in: [Manipulation](#) > [Class Attribute](#) | [CSS](#)

---

## **.toggleClass()**

Add or remove one or more classes from each element in the set of matched elements, depending on either the class's presence or the value of the switch argument.

Also in: [Forms](#) | [Manipulation](#) > [General Attributes](#)

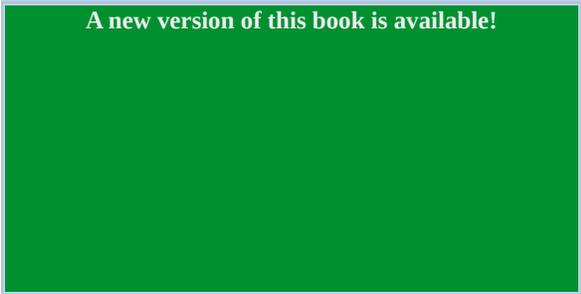
---

## `.val()`

Get the current value of the first element in the set of matched elements or set the value of every matched element.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

## Category: Callbacks Object

The `jQuery.Callbacks()` function, introduced in version 1.7, returns a multi-purpose object that provides a powerful way to manage callback lists. It supports adding, removing, firing, and disabling callbacks.

---

## **callbacks.add()**

Add a callback or a collection of callbacks to a callback list.

---

## **callbacks.disable()**

Disable a callback list from doing anything more.

---

## **callbacks.disabled()**

Determine if the callbacks list has been disabled.

---

## **callbacks.empty()**

Remove all of the callbacks from a list.

---

## **callbacks.fire()**

Call all of the callbacks with the given arguments

---

## **callbacks.fired()**

Determine if the callbacks have already been called at least once.

---

## **callbacks.fireWith()**

Call all callbacks in a list with the given context and arguments.

---

## callbacks.has()

Determine whether a supplied callback is in a list

---

## **callbacks.lock()**

Lock a callback list in its current state.

---

## **callbacks.locked()**

Determine if the callbacks list has been locked.

---

## **callbacks.remove()**

Remove a callback or a collection of callbacks from a callback list.

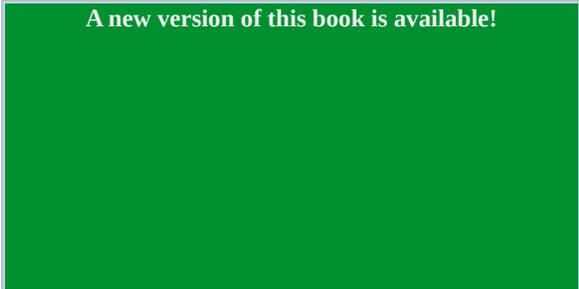
---

## jQuery.Callbacks()

A multi-purpose callbacks list object that provides a powerful way to manage callback lists.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

**Category: Core**

---

## jQuery()

Return a collection of matched elements either found in the DOM based on passed argument(s) or created by passing an HTML string.

---

## jQuery.holdReady()

Holds or releases the execution of jQuery's ready event.

Also in: [Miscellaneous](#) > [Setup Methods](#)

---

## jQuery.noConflict()

Relinquish jQuery's control of the \$ variable.

---

## jQuery.sub()

Creates a new copy of jQuery whose properties and methods can be modified without affecting the original jQuery object.

Also in: [Deferred Object](#)

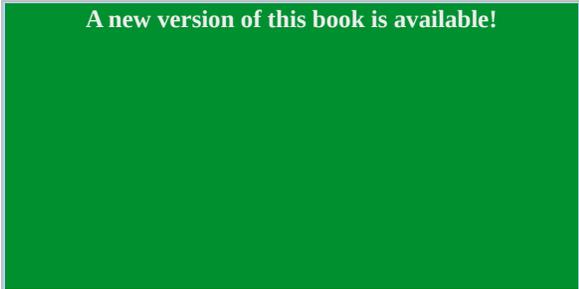
---

## jQuery.when()

Provides a way to execute callback functions based on one or more objects, usually Deferred objects that represent asynchronous events.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

# Category: CSS

These methods get and set CSS-related properties of elements.

Also in: [Attributes](#) | [Manipulation](#) > [Class Attribute](#)

---

## **.addClass()**

Adds the specified class(es) to each of the set of matched elements.

Also in: [Manipulation](#) > [Style Properties](#)

---

## `.css()`

Get the value of a style property for the first element in the set of matched elements or set one or more CSS properties for every matched element.

Also in: [Attributes](#) | [Manipulation](#) > [Class Attribute](#)

---

## **.hasClass()**

Determine whether any of the matched elements are assigned the given class.

Also in: [Dimensions](#) | [Manipulation](#) > [Style Properties](#)

---

## **.height()**

Get the current computed height for the first element in the set of matched elements or set the height of every matched element.

Also in: [Dimensions](#) | [Manipulation](#) > [Style Properties](#)

---

## **.innerHeight()**

Get the current computed height for the first element in the set of matched elements, including padding but not border.

Also in: [Dimensions](#) | [Manipulation](#) > [Style Properties](#)

---

## **.innerWidth()**

Get the current computed width for the first element in the set of matched elements, including padding but not border.

---

## jQuery.cssHooks

Hook directly into jQuery to override how particular CSS properties are retrieved or set, normalize CSS property naming, or create custom properties.

Also in: [Offset](#) | [Manipulation](#) > [Style Properties](#)

---

## **.offset()**

Get the current coordinates of the first element, or set the coordinates of every element, in the set of matched elements, relative to the document.

Also in: [Dimensions](#) | [Manipulation](#) > [Style Properties](#)

---

## **.outerHeight()**

Get the current computed height for the first element in the set of matched elements, including padding, border, and optionally margin. Returns an integer (without "px") representation of the value or null if called on an empty set of elements.

Also in: [Dimensions](#) | [Manipulation](#) > [Style Properties](#)

---

## `.outerWidth()`

Get the current computed width for the first element in the set of matched elements, including padding and border.

Also in: [Offset](#) | [Manipulation](#) > [Style Properties](#)

---

## **.position()**

Get the current coordinates of the first element in the set of matched elements, relative to the offset parent.

Also in: [Attributes](#) | [Manipulation](#) > [Class Attribute](#)

---

## `.removeClass()`

Remove a single class, multiple classes, or all classes from each element in the set of matched elements.

Also in: [Offset](#) | [Manipulation](#) > [Style Properties](#)

---

## **.scrollLeft()**

Get the current horizontal position of the scroll bar for the first element in the set of matched elements or set the horizontal position of the scroll bar for every matched element.

Also in: [Offset](#) | [Manipulation](#) > [Style Properties](#)

---

## **.scrollTop()**

Get the current vertical position of the scroll bar for the first element in the set of matched elements or set the vertical position of the scroll bar for every matched element.

Also in: [Attributes](#) | [Manipulation](#) > [Class Attribute](#)

---

## **.toggleClass()**

Add or remove one or more classes from each element in the set of matched elements, depending on either the class's presence or the value of the switch argument.

Also in: [Dimensions](#) | [Manipulation](#) > [Style Properties](#)

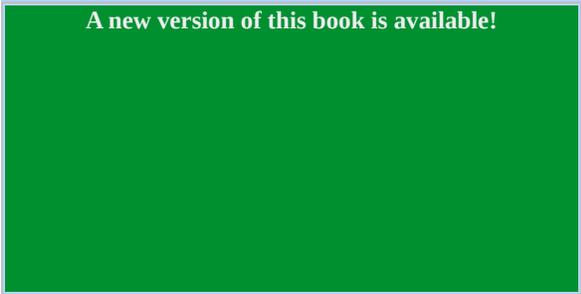
---

## **.width()**

Get the current computed width for the first element in the set of matched elements or set the width of every matched element.

POWERED BY **HERONOTE**

A new version of this book is available!



---

# Category: Data

These methods allow us to associate arbitrary data with specific DOM elements.

Also in: [Effects](#) > [Custom](#) | [Utilities](#)

---

## **.clearQueue()**

Remove from the queue all items that have not yet been run.

Also in: [Miscellaneous](#) > [Data Storage](#)

---

## **.data()**

Store arbitrary data associated with the matched elements or return the value at the named data store for the first element in the set of matched elements.

Also in: [Effects](#) > [Custom](#) | [Utilities](#)

---

## **.dequeue()**

Execute the next function on the queue for the matched elements.

Also in: [Utilities](#)

---

## jQuery.data()

Store arbitrary data associated with the specified element and/or return the value that was set.

Also in: [Utilities](#)

---

## jQuery.dequeue()

Execute the next function on the queue for the matched element.

---

## jQuery.hasData()

Determine whether an element has any jQuery data associated with it.

Also in: [Utilities](#)

---

## jQuery.queue()

Show or manipulate the queue of functions to be executed on the matched element.

Also in: [Utilities](#)

---

## jQuery.removeData()

Remove a previously-stored piece of data.

Also in: [Effects](#) > [Custom](#) | [Utilities](#)

---

## `.queue()`

Show or manipulate the queue of functions to be executed on the matched elements.

Also in: [Miscellaneous](#) > [Data Storage](#)

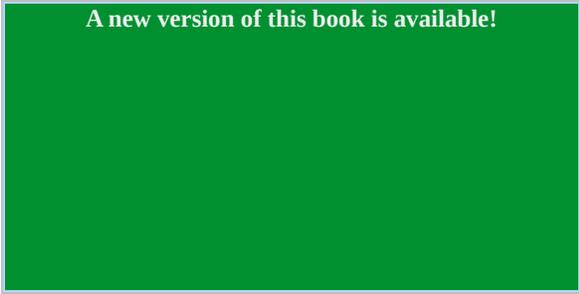
---

## **.removeData()**

Remove a previously-stored piece of data.

POWERED BY **HERONOTE**

A new version of this book is available!



---

## Category: Deferred Object

The Deferred object, introduced in jQuery 1.5, is a chainable utility object created by calling the `jQuery.Deferred()` method. It can register multiple callbacks into callback queues, invoke callback queues, and relay the success or failure state of any synchronous or asynchronous function.

The Deferred object is chainable, similar to the way a jQuery object is chainable, but it has its own methods. After creating a Deferred object, you can use any of the methods below by either chaining directly from the object creation or saving the object in a variable and invoking one or more methods on that variable.

---

## **deferred.always()**

Add handlers to be called when the Deferred object is either resolved or rejected.

---

## **deferred.done()**

Add handlers to be called when the Deferred object is resolved.

---

## **deferred.fail()**

Add handlers to be called when the Deferred object is rejected.

---

## **deferred.isRejected()**

Determine whether a Deferred object has been rejected.

---

## **deferred.isResolved()**

Determine whether a Deferred object has been resolved.

---

## **deferred.notify()**

Call the progressCallbacks on a Deferred object with the given args.

---

## **deferred.notifyWith()**

Call the progressCallbacks on a Deferred object with the given context and args.

---

## **deferred.pipe()**

Utility method to filter and/or chain Deferreds.

---

## **deferred.progress()**

Add handlers to be called when the Deferred object generates progress notifications.

---

## **deferred.promise()**

Return a Deferred's Promise object.

---

## **deferred.reject()**

Reject a Deferred object and call any failCallbacks with the given args.

---

## **deferred.rejectWith()**

Reject a Deferred object and call any failCallbacks with the given context and args.

---

## **deferred.resolve()**

Resolve a Deferred object and call any doneCallbacks with the given args.

---

## **deferred.resolveWith()**

Resolve a Deferred object and call any doneCallbacks with the given context and args.

---

## **deferred.state()**

Determine the current state of a Deferred object.

---

## **deferred.then()**

Add handlers to be called when the Deferred object is resolved, rejected, or still in progress.

---

## jQuery.Deferred()

A constructor function that returns a chainable utility object with methods to register multiple callbacks into callback queues, invoke callback queues, and relay the success or failure state of any synchronous or asynchronous function.

Also in: [Core](#)

---

## jQuery.when()

Provides a way to execute callback functions based on one or more objects, usually Deferred objects that represent asynchronous events.

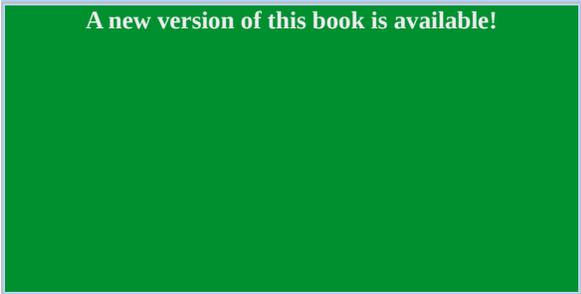
---

## **.promise()**

Return a Promise object to observe when all actions of a certain type bound to the collection, queued or not, have finished.

POWERED BY **HERONOTE**

A new version of this book is available!



---

# Category: Deprecated

Also in: [Deprecated](#) > [Deprecated 1.8](#) | [Events](#) > [Mouse Events](#)

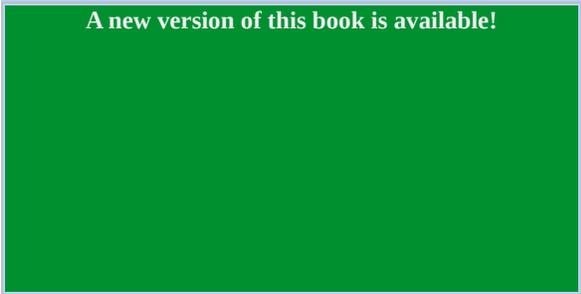
---

## **.toggle()**

Bind two or more handlers to the matched elements, to be executed on alternate clicks.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

## Category: Deprecated 1.8

All the aspects of the API that were deprecated in the corresponding version of jQuery.

For more information, see the Release Notes/Changelog at <http://blog.jquery.com/2012/08/09/jquery-1-8-released/>

Also in: [Events](#) > [Mouse Events](#)

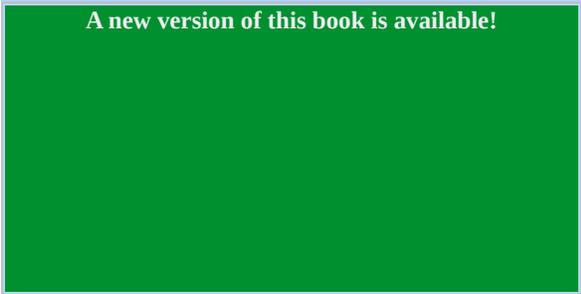
---

## **.toggle()**

Bind two or more handlers to the matched elements, to be executed on alternate clicks.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

# Category: Dimensions

These methods are used to get and set the CSS dimensions for the various properties.

Also in: [CSS](#) | [Manipulation](#) > [Style Properties](#)

---

## **.height()**

Get the current computed height for the first element in the set of matched elements or set the height of every matched element.

Also in: [CSS](#) | [Manipulation](#) > [Style Properties](#)

---

## **.innerHeight()**

Get the current computed height for the first element in the set of matched elements, including padding but not border.

Also in: [CSS](#) | [Manipulation](#) > [Style Properties](#)

---

## **.innerWidth()**

Get the current computed width for the first element in the set of matched elements, including padding but not border.

Also in: [CSS](#) | [Manipulation](#) > [Style Properties](#)

---

## **.outerHeight()**

Get the current computed height for the first element in the set of matched elements, including padding, border, and optionally margin. Returns an integer (without "px") representation of the value or null if called on an empty set of elements.

Also in: [CSS](#) | [Manipulation](#) > [Style Properties](#)

---

## `.outerWidth()`

Get the current computed width for the first element in the set of matched elements, including padding and border.

Also in: [CSS](#) | [Manipulation](#) > [Style Properties](#)

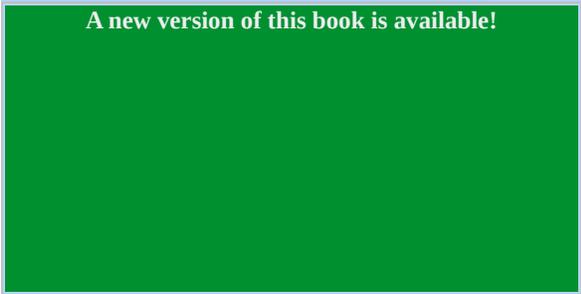
---

## **.width()**

Get the current computed width for the first element in the set of matched elements or set the width of every matched element.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

# Category: Effects

The jQuery library provides several techniques for adding animation to a web page. These include simple, standard animations that are frequently used, and the ability to craft sophisticated custom effects.

Also in: [Effects](#) > [Custom](#)

---

## **.animate()**

Perform a custom animation of a set of CSS properties.

Also in: [Effects](#) > [Custom](#) | [Data](#) | [Utilities](#)

---

## **.clearQueue()**

Remove from the queue all items that have not yet been run.

Also in: [Effects](#) > [Custom](#)

---

## `.delay()`

Set a timer to delay execution of subsequent items in the queue.

Also in: [Effects > Custom](#) | [Data](#) | [Utilities](#)

---

## **.dequeue()**

Execute the next function on the queue for the matched elements.

Also in: [Effects](#) > [Fading](#)

---

## **.fadeIn()**

Display the matched elements by fading them to opaque.

Also in: [Effects > Fading](#)

---

## **.fadeOut()**

Hide the matched elements by fading them to transparent.

Also in: [Effects](#) > [Fading](#)

---

## **.fadeTo()**

Adjust the opacity of the matched elements.

Also in: [Effects](#) > [Fading](#)

---

## **.fadeToggle()**

Display or hide the matched elements by animating their opacity.

Also in: [Effects](#) > [Custom](#)

---

## **.finish()**

Stop the currently-running animation, remove all queued animations, and complete all animations for the matched elements.

Also in: [Effects](#) > [Basics](#)

---

## **.hide()**

Hide the matched elements.

Also in: [Effects > Custom](#) | [Properties > Properties of the Global jQuery Object](#)

---

## jQuery.fx.interval

The rate (in milliseconds) at which animations fire.

Also in: [Effects > Custom](#) | [Properties > Properties of the Global jQuery Object](#)

---

## jQuery.fx.off

Globally disable all animations.

Also in: [Effects](#) > [Custom](#) | [Data](#) | [Utilities](#)

---

## `.queue()`

Show or manipulate the queue of functions to be executed on the matched elements.

Also in: [Effects](#) > [Basics](#)

---

## **.show()**

Display the matched elements.

Also in: [Effects](#) > [Sliding](#)

---

## **.slideDown()**

Display the matched elements with a sliding motion.

Also in: [Effects](#) > [Sliding](#)

---

## **.slideToggle()**

Display or hide the matched elements with a sliding motion.

Also in: [Effects](#) > [Sliding](#)

---

## `.slideUp()`

Hide the matched elements with a sliding motion.

Also in: [Effects](#) > [Custom](#)

---

## **.stop()**

Stop the currently-running animation on the matched elements.

Also in: [Effects](#) > [Basics](#)

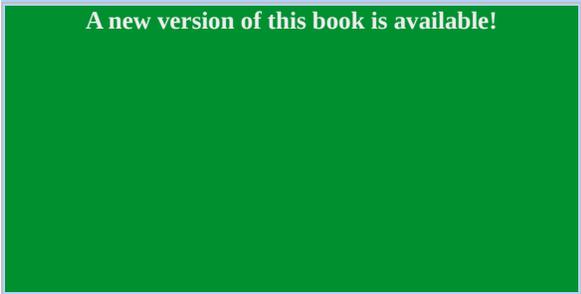
---

**.toggle()**

Display or hide the matched elements.

POWERED BY **HERONOTE**

A new version of this book is available!



---

## Category: Basics

---

## **.hide()**

Hide the matched elements.

---

**.show()**

Display the matched elements.

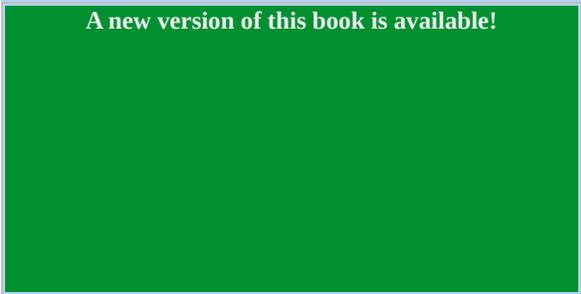
---

**.toggle()**

Display or hide the matched elements.

POWERED BY **HERONOTE**

A new version of this book is available!



---

## Category: Custom

These methods allow you to create effects that are not provided “out of the box” by jQuery.

---

## **.animate()**

Perform a custom animation of a set of CSS properties.

Also in: [Data](#) | [Utilities](#)

---

## **.clearQueue()**

Remove from the queue all items that have not yet been run.

---

## **.delay()**

Set a timer to delay execution of subsequent items in the queue.

Also in: [Data](#) | [Utilities](#)

---

## **.dequeue()**

Execute the next function on the queue for the matched elements.

---

## **.finish()**

Stop the currently-running animation, remove all queued animations, and complete all animations for the matched elements.

Also in: [Properties](#) > [Properties of the Global jQuery Object](#)

---

## jQuery.fx.interval

The rate (in milliseconds) at which animations fire.

Also in: [Properties](#) > [Properties of the Global jQuery Object](#)

---

## jQuery.fx.off

Globally disable all animations.

Also in: [Data](#) | [Utilities](#)

---

## **.queue()**

Show or manipulate the queue of functions to be executed on the matched elements.

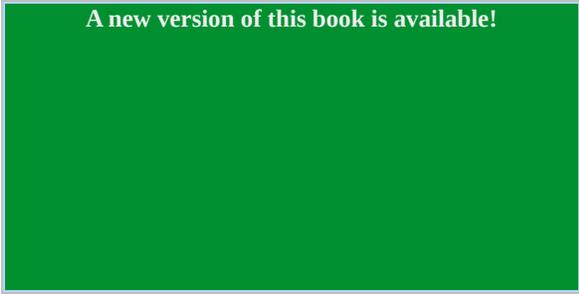
---

## **.stop()**

Stop the currently-running animation on the matched elements.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

## Category: Fading

These methods adjust the opacity of elements.

---

## **.fadeIn()**

Display the matched elements by fading them to opaque.

---

## **.fadeOut()**

Hide the matched elements by fading them to transparent.

---

## **.fadeTo()**

Adjust the opacity of the matched elements.

Also in: [Effects](#)

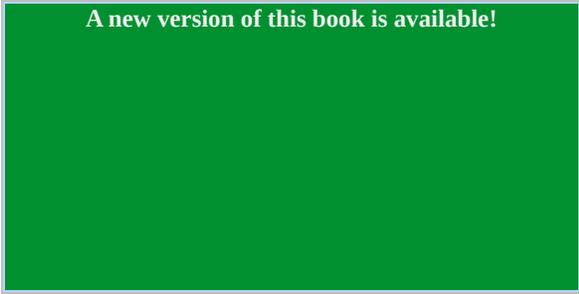
---

## **.fadeToggle()**

Display or hide the matched elements by animating their opacity.

POWERED BY **HERONOTE**

A new version of this book is available!



---

## **Category: Sliding**

---

## **.slideDown()**

Display the matched elements with a sliding motion.

---

## **.slideToggle()**

Display or hide the matched elements with a sliding motion.

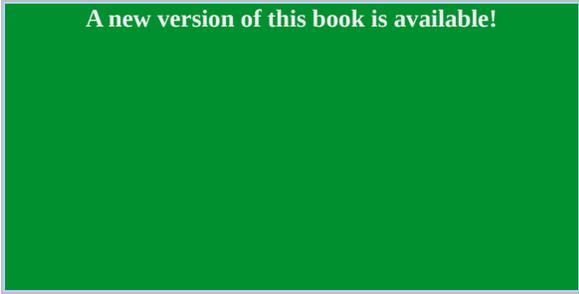
---

## `.slideUp()`

Hide the matched elements with a sliding motion.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

# Category: Events

These methods are used to register behaviors to take effect when the user interacts with the browser, and to further manipulate those registered behaviors.

Also in: [Events](#) > [Event Handler Attachment](#)

---

## **.bind()**

Attach a handler to an event for the elements.

Also in: [Events > Form Events](#) | [Forms](#)

---

## **.blur()**

Bind an event handler to the “blur” JavaScript event, or trigger that event on an element.

Also in: [Events > Form Events](#) | [Forms](#)

---

## `.change()`

Bind an event handler to the “change” JavaScript event, or trigger that event on an element.

Also in: [Events](#) > [Mouse Events](#)

---

## `.click()`

Bind an event handler to the “click” JavaScript event, or trigger that event on an element.

Also in: [Events](#) > [Mouse Events](#)

---

## **.dblclick()**

Bind an event handler to the “dblclick” JavaScript event, or trigger that event on an element.

Also in: [Events](#) > [Event Handler Attachment](#)

---

## **.delegate()**

Attach a handler to one or more events for all elements that match the selector, now or in the future, based on a specific set of root elements.

Also in: [Events](#) > [Event Handler Attachment](#)

---

## **.die()**

Remove event handlers previously attached using `.live()` from the elements.

Also in: [Events > Browser Events](#)

---

## **.error()**

Bind an event handler to the “error” JavaScript event.

Also in: [Events](#) > [Event Object](#)

---

## **event.currentTarget**

The current DOM element within the event bubbling phase.

Also in: [Events](#) > [Event Object](#)

---

## **event.data**

An optional object of data passed to an event method when the current executing handler is bound.

Also in: [Events > Event Object](#)

---

## event.delegateTarget

The element where the currently-called jQuery event handler was attached.

Also in: [Events](#) > [Event Object](#)

---

## **event.isDefaultPrevented()**

Returns whether `event.preventDefault()` was ever called on this event object.

Also in: [Events](#) > [Event Object](#)

---

## **event.isImmediatePropagationStopped()**

Returns whether `event.stopImmediatePropagation()` was ever called on this event object.

Also in: [Events](#) > [Event Object](#)

---

## **event.isPropagationStopped()**

Returns whether `event.stopPropagation()` was ever called on this event object.

Also in: [Events](#) > [Event Object](#)

---

## event.metaKey

Indicates whether the META key was pressed when the event fired.

Also in: [Events](#) > [Event Object](#)

---

## event.namespace

The namespace specified when the event was triggered.

Also in: [Events](#) > [Event Object](#)

---

## event.pageX

The mouse position relative to the left edge of the document.

Also in: [Events](#) > [Event Object](#)

---

## event.pageY

The mouse position relative to the top edge of the document.

Also in: [Events](#) > [Event Object](#)

---

## **event.preventDefault()**

If this method is called, the default action of the event will not be triggered.

Also in: [Events > Event Object](#)

---

## event.relatedTarget

The other DOM element involved in the event, if any.

Also in: [Events](#) > [Event Object](#)

---

## **event.result**

The last value returned by an event handler that was triggered by this event, unless the value was undefined.

Also in: [Events > Event Object](#)

---

## **event.stopImmediatePropagation()**

Keeps the rest of the handlers from being executed and prevents the event from bubbling up the DOM tree.

Also in: [Events](#) > [Event Object](#)

---

## **event.stopPropagation()**

Prevents the event from bubbling up the DOM tree, preventing any parent handlers from being notified of the event.

Also in: [Events](#) > [Event Object](#)

---

## event.target

The DOM element that initiated the event.

Also in: [Events](#) > [Event Object](#)

---

## event.timeStamp

The difference in milliseconds between the time the browser created the event and January 1, 1970.

Also in: [Events](#) > [Event Object](#)

---

## event.type

Describes the nature of the event.

Also in: [Events](#) > [Event Object](#)

---

## event.which

For key or mouse events, this property indicates the specific key or button that was pressed.

Also in: [Events > Form Events](#) | [Forms](#)

---

## **.focus()**

Bind an event handler to the “focus” JavaScript event, or trigger that event on an element.

Also in: [Events > Keyboard Events](#) | [Events > Mouse Events](#)

---

## **.focusin()**

Bind an event handler to the “focusin” event.

Also in: [Events > Keyboard Events](#) | [Events > Mouse Events](#)

---

## **.focusout()**

Bind an event handler to the “focusout” JavaScript event.

Also in: [Events](#) > [Mouse Events](#)

---

## **.hover()**

Bind one or two handlers to the matched elements, to be executed when the mouse pointer enters and leaves the elements.

Also in: [Events](#) > [Event Handler Attachment](#) | [Utilities](#)

---

## jQuery.proxy()

Takes a function and returns a new one that will always have a particular context.

Also in: [Events](#) > [Keyboard Events](#)

---

## **.keydown()**

Bind an event handler to the “keydown” JavaScript event, or trigger that event on an element.

Also in: [Events](#) > [Keyboard Events](#)

---

## **.keypress()**

Bind an event handler to the “keypress” JavaScript event, or trigger that event on an element.

Also in: [Events](#) > [Keyboard Events](#)

---

## **.keyup()**

Bind an event handler to the “keyup” JavaScript event, or trigger that event on an element.

Also in: [Events](#) > [Event Handler Attachment](#)

---

## `.live()`

Attach an event handler for all elements which match the current selector, now and in the future.

Also in: [Events](#) > [Document Loading](#)

---

## **.load()**

Bind an event handler to the “load” JavaScript event.

Also in: [Events](#) > [Mouse Events](#)

---

## **.mousedown()**

Bind an event handler to the “mousedown” JavaScript event, or trigger that event on an element.

Also in: [Events](#) > [Mouse Events](#)

---

## **.mouseenter()**

Bind an event handler to be fired when the mouse enters an element, or trigger that handler on an element.

Also in: [Events](#) > [Mouse Events](#)

---

## **.mouseleave()**

Bind an event handler to be fired when the mouse leaves an element, or trigger that handler on an element.

Also in: [Events](#) > [Mouse Events](#)

---

## `.mousemove()`

Bind an event handler to the “mousemove” JavaScript event, or trigger that event on an element.

Also in: [Events](#) > [Mouse Events](#)

---

## **.mouseout()**

Bind an event handler to the “mouseout” JavaScript event, or trigger that event on an element.

Also in: [Events](#) > [Mouse Events](#)

---

## `.mouseover()`

Bind an event handler to the “mouseover” JavaScript event, or trigger that event on an element.

Also in: [Events](#) > [Mouse Events](#)

---

## `.mouseup()`

Bind an event handler to the “mouseup” JavaScript event, or trigger that event on an element.

Also in: [Events](#) > [Event Handler Attachment](#)

---

## **.off()**

Remove an event handler.

Also in: [Events](#) > [Event Handler Attachment](#)

---

## `.on()`

Attach an event handler function for one or more events to the selected elements.

Also in: [Events](#) > [Event Handler Attachment](#)

---

## **.one()**

Attach a handler to an event for the elements. The handler is executed at most once per element.

Also in: [Events](#) > [Document Loading](#)

---

## **.ready()**

Specify a function to execute when the DOM is fully loaded.

Also in: [Events](#) > [Browser Events](#)

---

## **.resize()**

Bind an event handler to the “resize” JavaScript event, or trigger that event on an element.

Also in: [Events](#) > [Browser Events](#)

---

## **.scroll()**

Bind an event handler to the “scroll” JavaScript event, or trigger that event on an element.

Also in: [Events > Form Events](#) | [Forms](#)

---

## **.select()**

Bind an event handler to the “select” JavaScript event, or trigger that event on an element.

Also in: [Events > Form Events](#) | [Forms](#)

---

## **.submit()**

Bind an event handler to the “submit” JavaScript event, or trigger that event on an element.

Also in: [Deprecated](#) > [Deprecated 1.8](#) | [Events](#) > [Mouse Events](#)

---

## **.toggle()**

Bind two or more handlers to the matched elements, to be executed on alternate clicks.

Also in: [Events](#) > [Event Handler Attachment](#)

---

## **.trigger()**

Execute all handlers and behaviors attached to the matched elements for the given event type.

Also in: [Events](#) > [Event Handler Attachment](#)

---

## **.triggerHandler()**

Execute all handlers attached to an element for an event.

Also in: [Events](#) > [Event Handler Attachment](#)

---

## **.unbind()**

Remove a previously-attached event handler from the elements.

Also in: [Events](#) > [Event Handler Attachment](#)

---

## **.undelegate()**

Remove a handler from the event for all elements which match the current selector, based upon a specific set of root elements.

Also in: [Events](#) > [Document Loading](#)

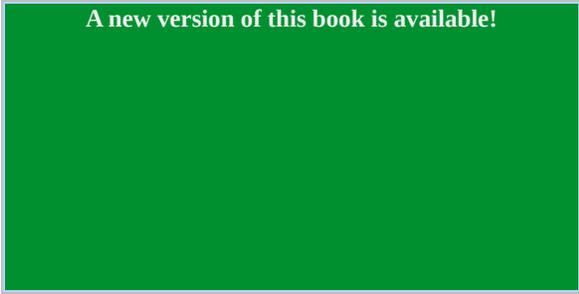
---

## **.unload()**

Bind an event handler to the “unload” JavaScript event.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

# Category: Browser Events

---

## **.error()**

Bind an event handler to the “error” JavaScript event.

---

## **.resize()**

Bind an event handler to the “resize” JavaScript event, or trigger that event on an element.

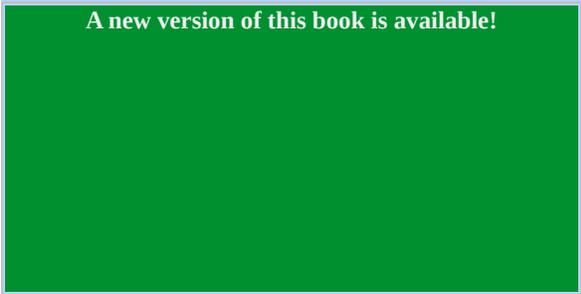
---

## **.scroll()**

Bind an event handler to the “scroll” JavaScript event, or trigger that event on an element.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

## Category: Document Loading

---

## **.load()**

Bind an event handler to the “load” JavaScript event.

---

## **.ready()**

Specify a function to execute when the DOM is fully loaded.

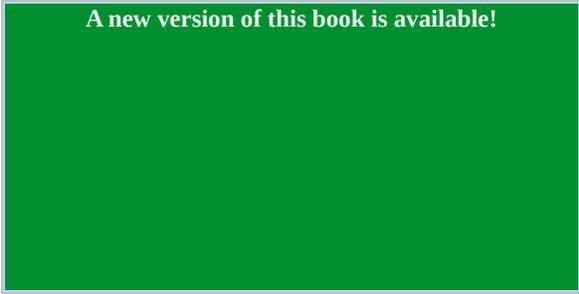
---

## **.unload()**

Bind an event handler to the “unload” JavaScript event.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

## **Category: Event Handler Attachment**

---

## **.bind()**

Attach a handler to an event for the elements.

---

## **.delegate()**

Attach a handler to one or more events for all elements that match the selector, now or in the future, based on a specific set of root elements.

---

## **.die()**

Remove event handlers previously attached using `.live()` from the elements.

Also in: [Utilities](#)

---

## jQuery.proxy()

Takes a function and returns a new one that will always have a particular context.

---

## **.live()**

Attach an event handler for all elements which match the current selector, now and in the future.

---

**.off()**

Remove an event handler.

---

## **.on()**

Attach an event handler function for one or more events to the selected elements.

---

## **.one()**

Attach a handler to an event for the elements. The handler is executed at most once per element.

---

## **.trigger()**

Execute all handlers and behaviors attached to the matched elements for the given event type.

---

## **.triggerHandler()**

Execute all handlers attached to an element for an event.

---

## **.unbind()**

Remove a previously-attached event handler from the elements.

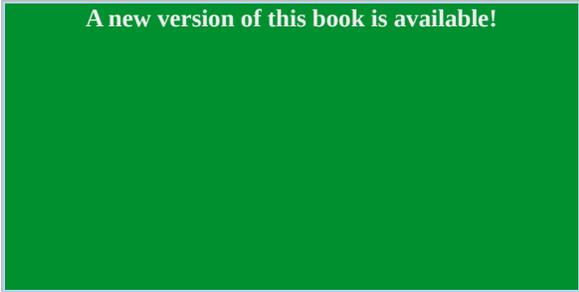
---

## **.undelegate()**

Remove a handler from the event for all elements which match the current selector, based upon a specific set of root elements.

POWERED BY [HERONOTE](#)

A new version of this book is available!



# Category: Event Object

jQuery's event system normalizes the event object according to [W3C standards](#). The event object is guaranteed to be passed to the event handler. Most properties from the original event are copied over and normalized to the new event object.

## jQuery.Event Constructor

The `jQuery.Event` constructor is exposed and can be used when calling [trigger](#). The `new` operator is optional.

Check [trigger](#)'s documentation to see how to combine it with your own event object.

Example:

```
//Create a new jQuery.Event object without
the "new" operator.
var e = jQuery.Event("click");

// trigger an artificial click event
jQuery("body").trigger( e );
```

As of jQuery 1.6, you can also pass an object to `jQuery.Event()` and its properties will be set on the newly created Event object.

Example:

```
// Create a new jQuery.Event object with
specified event properties.
var e = jQuery.Event("keydown", { keyCode:
64 });

// trigger an artificial keydown event
```

```
with keyCode 64
  jQuery("body").trigger( e );
```

## Event Properties

jQuery normalizes the following properties for cross-browser consistency:

target

relatedTarget

pageX

pageY

which

metaKey

The following properties are also copied to the event object, though some of their values may be undefined depending on the event:

altKey, bubbles, button, cancelable, charCode, clientX, clientY, ctrlKey, currentTarget, data, detail, eventPhase, metaKey, offsetX, offsetY, originalTarget, pageX, pageY, prevValue, relatedTarget, screenX, screenY, shiftKey, target, view, which

## Other Properties

Certain events may have properties specific to them. Those can be accessed as properties of the `event.originalEvent` object. To make special properties available in all event objects, they can be added to the `jQuery.event.props` array. This is not recommended, since it adds overhead to every event delivered by jQuery.

Example:

```
// add the dataTransfer property for use
with the native `drop` event
// to capture information about files
dropped into the browser window
```

```
jQuery.event.props.push("dataTransfer");
```

---

## **event.currentTarget**

The current DOM element within the event bubbling phase.

---

## **event.data**

An optional object of data passed to an event method when the current executing handler is bound.

Also in: [Events](#)

---

## **event.delegateTarget**

The element where the currently-called jQuery event handler was attached.

---

## **event.isDefaultPrevented()**

Returns whether `event.preventDefault()` was ever called on this event object.

---

## **event.isImmediatePropagationStopped()**

Returns whether `event.stopImmediatePropagation()` was ever called on this event object.

---

## **event.isPropagationStopped()**

Returns whether `event.stopPropagation()` was ever called on this event object.

---

## **event.metaKey**

Indicates whether the META key was pressed when the event fired.

---

## **event.namespace**

The namespace specified when the event was triggered.

---

## **event.pageX**

The mouse position relative to the left edge of the document.

---

## **event.pageY**

The mouse position relative to the top edge of the document.

---

## **event.preventDefault()**

If this method is called, the default action of the event will not be triggered.

---

## **event.relatedTarget**

The other DOM element involved in the event, if any.

---

## **event.result**

The last value returned by an event handler that was triggered by this event, unless the value was undefined.

---

## **event.stopImmediatePropagation()**

Keeps the rest of the handlers from being executed and prevents the event from bubbling up the DOM tree.

---

## **event.stopPropagation()**

Prevents the event from bubbling up the DOM tree, preventing any parent handlers from being notified of the event.

---

## **event.target**

The DOM element that initiated the event.

---

## **event.timeStamp**

The difference in milliseconds between the time the browser created the event and January 1, 1970.

---

## **event.type**

Describes the nature of the event.

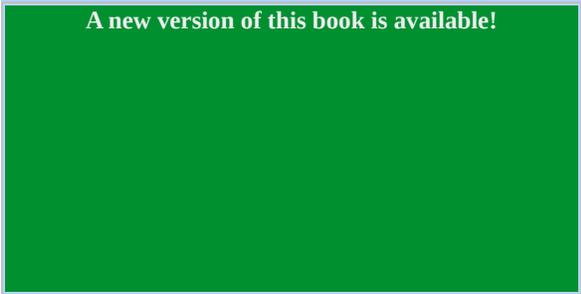
---

## event.which

For key or mouse events, this property indicates the specific key or button that was pressed.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

# Category: Form Events

Also in: [Forms](#)

---

## **.blur()**

Bind an event handler to the “blur” JavaScript event, or trigger that event on an element.

Also in: [Forms](#)

---

## **.change()**

Bind an event handler to the “change” JavaScript event, or trigger that event on an element.

Also in: [Forms](#)

---

## **.focus()**

Bind an event handler to the “focus” JavaScript event, or trigger that event on an element.

Also in: [Forms](#)

---

## **.select()**

Bind an event handler to the “select” JavaScript event, or trigger that event on an element.

Also in: [Forms](#)

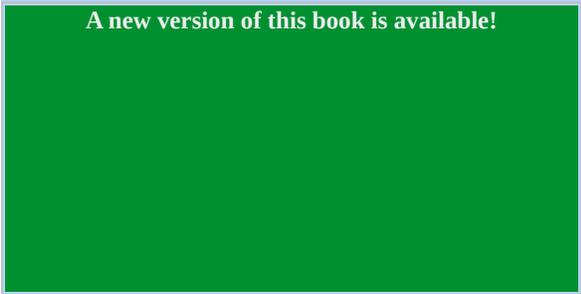
---

## **.submit()**

Bind an event handler to the “submit” JavaScript event, or trigger that event on an element.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

# Category: Keyboard Events

Also in: [Events](#) > [Mouse Events](#)

---

## **.focusin()**

Bind an event handler to the “focusin” event.

Also in: [Events](#) > [Mouse Events](#)

---

## **.focusout()**

Bind an event handler to the “focusout” JavaScript event.

---

## **.keydown()**

Bind an event handler to the “keydown” JavaScript event, or trigger that event on an element.

---

## **.keypress()**

Bind an event handler to the “keypress” JavaScript event, or trigger that event on an element.

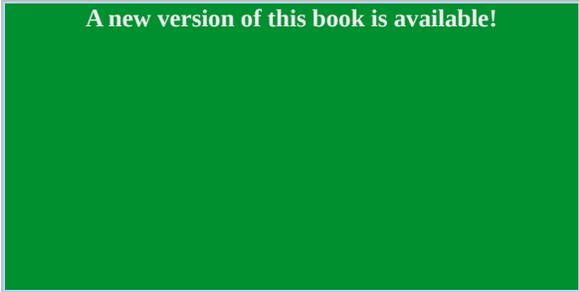
---

## **.keyup()**

Bind an event handler to the “keyup” JavaScript event, or trigger that event on an element.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

## Category: Mouse Events

---

## **.click()**

Bind an event handler to the “click” JavaScript event, or trigger that event on an element.

---

## **.dblclick()**

Bind an event handler to the “dblclick” JavaScript event, or trigger that event on an element.

Also in: [Events](#) > [Keyboard Events](#)

---

## **.focusin()**

Bind an event handler to the “focusin” event.

Also in: [Events](#) > [Keyboard Events](#)

---

## **.focusout()**

Bind an event handler to the “focusout” JavaScript event.

---

## **.hover()**

Bind one or two handlers to the matched elements, to be executed when the mouse pointer enters and leaves the elements.

---

## **.mousedown()**

Bind an event handler to the “mousedown” JavaScript event, or trigger that event on an element.

---

## **.mouseenter()**

Bind an event handler to be fired when the mouse enters an element, or trigger that handler on an element.

---

## **.mouseleave()**

Bind an event handler to be fired when the mouse leaves an element, or trigger that handler on an element.

---

## **.mousemove()**

Bind an event handler to the “mousemove” JavaScript event, or trigger that event on an element.

---

## **.mouseout()**

Bind an event handler to the “mouseout” JavaScript event, or trigger that event on an element.

---

## **.mouseover()**

Bind an event handler to the “mouseover” JavaScript event, or trigger that event on an element.

---

## `.mouseup()`

Bind an event handler to the “mouseup” JavaScript event, or trigger that event on an element.

Also in: [Deprecated](#) > [Deprecated 1.8](#)

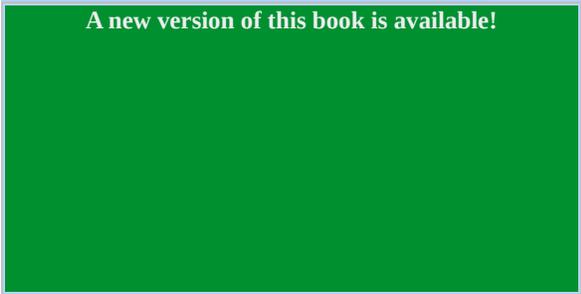
---

## **.toggle()**

Bind two or more handlers to the matched elements, to be executed on alternate clicks.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

# Category: Forms

These methods and event handlers handle forms and their various elements.

Also in: [Events](#) > [Form Events](#)

---

## **.blur()**

Bind an event handler to the “blur” JavaScript event, or trigger that event on an element.

Also in: [Events](#) > [Form Events](#)

---

## **.change()**

Bind an event handler to the “change” JavaScript event, or trigger that event on an element.

Also in: [Events](#) > [Form Events](#)

---

## **.focus()**

Bind an event handler to the “focus” JavaScript event, or trigger that event on an element.

Also in: [Miscellaneous](#) > [Collection Manipulation](#) | [Ajax](#) > [Helper Functions](#)

---

## jQuery.param()

Create a serialized representation of an array or object, suitable for use in a URL query string or Ajax request.

Also in: [Events](#) > [Form Events](#)

---

## **.select()**

Bind an event handler to the “select” JavaScript event, or trigger that event on an element.

Also in: [Ajax > Helper Functions](#)

---

## **.serialize()**

Encode a set of form elements as a string for submission.

Also in: [Ajax > Helper Functions](#)

---

## **.serializeArray()**

Encode a set of form elements as an array of names and values.

Also in: [Events](#) > [Form Events](#)

---

## **.submit()**

Bind an event handler to the “submit” JavaScript event, or trigger that event on an element.

Also in: [Attributes](#) | [Manipulation](#) > [General Attributes](#)

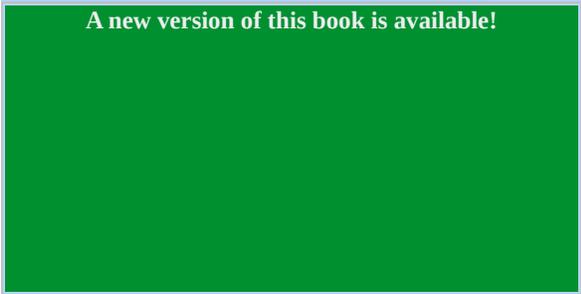
---

## `.val()`

Get the current value of the first element in the set of matched elements or set the value of every matched element.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

# Category: Internals

Although this category is referred to as 'internal', any methods documented within the API site should be considered public and may be freely used.

Also in: [Properties](#) > [Properties of jQuery Object Instances](#)

---

## **.context**

The DOM node context originally passed to `jQuery()`; if none was passed then context will likely be the document.

Also in: [Properties](#) > [Properties of jQuery Object Instances](#)

---

## **.jquery**

A string containing the jQuery version number.

---

## jQuery.error()

Takes a string and throws an exception containing it.

---

## **.pushStack()**

Add a collection of DOM elements onto the jQuery stack.

Also in: [Properties](#) > [Properties of the Global jQuery Object](#)

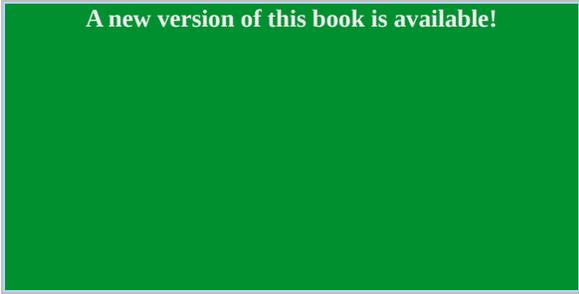
---

## **.selector**

A selector representing selector originally passed to jQuery().

POWERED BY **HERONOTE**

A new version of this book is available!



---

# Category: Manipulation

All of the methods in this section manipulate the DOM in some manner. A few of them simply change one of the attributes of an element (also listed in the [Attributes category](#)), while others set an element's style properties (also listed in the [CSS category](#)). Still others modify entire elements (or groups of elements) themselves—inserting, copying, removing, and so on. All of these methods are referred to as “setters,” as they change the values of properties.

A few of these methods—such as `.attr()`, `.html()`, and `.val()`—also act as “getters,” retrieving information from DOM elements for later use.

Also in: [Attributes](#) | [Manipulation > Class Attribute](#) | [CSS](#)

---

## **.addClass()**

Adds the specified class(es) to each of the set of matched elements.

Also in: [Manipulation > DOM Insertion, Outside](#)

---

## **.after()**

Insert content, specified by the parameter, after each element in the set of matched elements.

Also in: [Manipulation](#) > [DOM Insertion, Inside](#)

---

## **.append()**

Insert content, specified by the parameter, to the end of each element in the set of matched elements.

Also in: [Manipulation](#) > [DOM Insertion, Inside](#)

---

## **.appendTo()**

Insert every element in the set of matched elements to the end of the target.

Also in: [Attributes](#) | [Manipulation](#) > [General Attributes](#)

---

## **.attr()**

Get the value of an attribute for the first element in the set of matched elements or set one or more attributes for every matched element.

Also in: [Manipulation](#) > [DOM Insertion, Outside](#)

---

## **.before()**

Insert content, specified by the parameter, before each element in the set of matched elements.

Also in: [Manipulation](#) > [Copying](#)

---

## `.clone()`

Create a deep copy of the set of matched elements.

Also in: [CSS](#) | [Manipulation](#) > [Style Properties](#)

---

## `.css()`

Get the value of a style property for the first element in the set of matched elements or set one or more CSS properties for every matched element.

Also in: [Manipulation](#) > [DOM Removal](#)

---

## `.detach()`

Remove the set of matched elements from the DOM.

Also in: [Manipulation](#) > [DOM Removal](#)

---

## `.empty()`

Remove all child nodes of the set of matched elements from the DOM.

Also in: [Attributes](#) | [Manipulation > Class Attribute](#) | [CSS](#)

---

## **.hasClass()**

Determine whether any of the matched elements are assigned the given class.

Also in: [CSS](#) | [Dimensions](#) | [Manipulation > Style Properties](#)

---

## **.height()**

Get the current computed height for the first element in the set of matched elements or set the height of every matched element.

Also in: [Attributes](#) | [Manipulation](#) > [DOM Insertion, Inside](#)

---

## `.html()`

Get the HTML contents of the first element in the set of matched elements or set the HTML contents of every matched element.

Also in: [CSS](#) | [Dimensions](#) | [Manipulation > Style Properties](#)

---

## **.innerHeight()**

Get the current computed height for the first element in the set of matched elements, including padding but not border.

Also in: [CSS](#) | [Dimensions](#) | [Manipulation > Style Properties](#)

---

## **.innerWidth()**

Get the current computed width for the first element in the set of matched elements, including padding but not border.

Also in: [Manipulation > DOM Insertion, Outside](#)

---

## **.insertAfter()**

Insert every element in the set of matched elements after the target.

Also in: [Manipulation > DOM Insertion, Outside](#)

---

## **.insertBefore()**

Insert every element in the set of matched elements before the target.

Also in: [CSS](#) | [Offset](#) | [Manipulation > Style Properties](#)

---

## **.offset()**

Get the current coordinates of the first element, or set the coordinates of every element, in the set of matched elements, relative to the document.

Also in: [CSS](#) | [Dimensions](#) | [Manipulation](#) > [Style Properties](#)

---

## **.outerHeight()**

Get the current computed height for the first element in the set of matched elements, including padding, border, and optionally margin. Returns an integer (without “px”) representation of the value or null if called on an empty set of elements.

Also in: [CSS](#) | [Dimensions](#) | [Manipulation](#) > [Style Properties](#)

---

## `.outerWidth()`

Get the current computed width for the first element in the set of matched elements, including padding and border.

Also in: [CSS](#) | [Offset](#) | [Manipulation](#) > [Style Properties](#)

---

## **.position()**

Get the current coordinates of the first element in the set of matched elements, relative to the offset parent.

Also in: [Manipulation](#) > [DOM Insertion, Inside](#)

---

## **.prepend()**

Insert content, specified by the parameter, to the beginning of each element in the set of matched elements.

Also in: [Manipulation](#) > [DOM Insertion, Inside](#)

---

## **.prependTo()**

Insert every element in the set of matched elements to the beginning of the target.

Also in: [Attributes](#) | [Manipulation](#) > [General Attributes](#)

---

## **.prop()**

Get the value of a property for the first element in the set of matched elements or set one or more properties for every matched element.

Also in: [Manipulation](#) > [DOM Removal](#)

---

## `.remove()`

Remove the set of matched elements from the DOM.

Also in: [Attributes](#) | [Manipulation](#) > [General Attributes](#)

---

## `.removeAttr()`

Remove an attribute from each element in the set of matched elements.

Also in: [Attributes](#) | [Manipulation > Class Attribute](#) | [CSS](#)

---

## **.removeClass()**

Remove a single class, multiple classes, or all classes from each element in the set of matched elements.

Also in: [Attributes](#) | [Manipulation](#) > [General Attributes](#)

---

## **.removeProp()**

Remove a property for the set of matched elements.

Also in: [Manipulation](#) > [DOM Replacement](#)

---

## `.replaceAll()`

Replace each target element with the set of matched elements.

Also in: [Manipulation](#) > [DOM Replacement](#)

---

## `.replaceWith()`

Replace each element in the set of matched elements with the provided new content and return the set of elements that was removed.

Also in: [CSS](#) | [Offset](#) | [Manipulation](#) > [Style Properties](#)

---

## `.scrollLeft()`

Get the current horizontal position of the scroll bar for the first element in the set of matched elements or set the horizontal position of the scroll bar for every matched element.

Also in: [CSS](#) | [Offset](#) | [Manipulation](#) > [Style Properties](#)

---

## `.scrollTop()`

Get the current vertical position of the scroll bar for the first element in the set of matched elements or set the vertical position of the scroll bar for every matched element.

Also in: [Manipulation](#) > [DOM Insertion, Inside](#)

---

## **.text()**

Get the combined text contents of each element in the set of matched elements, including their descendants, or set the text contents of the matched elements.

Also in: [Attributes](#) | [Manipulation > Class Attribute](#) | [CSS](#)

---

## **.toggleClass()**

Add or remove one or more classes from each element in the set of matched elements, depending on either the class's presence or the value of the switch argument.

Also in: [Manipulation > DOM Insertion, Around](#) | [Manipulation > DOM Removal](#)

---

## `.unwrap()`

Remove the parents of the set of matched elements from the DOM, leaving the matched elements in their place.

Also in: [Attributes](#) | [Forms](#) | [Manipulation](#) > [General Attributes](#)

---

## `.val()`

Get the current value of the first element in the set of matched elements or set the value of every matched element.

Also in: [CSS](#) | [Dimensions](#) | [Manipulation > Style Properties](#)

---

## **.width()**

Get the current computed width for the first element in the set of matched elements or set the width of every matched element.

Also in: [Manipulation](#) > [DOM Insertion, Around](#)

---

## `.wrap()`

Wrap an HTML structure around each element in the set of matched elements.

Also in: [Manipulation](#) > [DOM Insertion, Around](#)

---

## **.wrapAll()**

Wrap an HTML structure around all elements in the set of matched elements.

Also in: [Manipulation](#) > [DOM Insertion, Around](#)

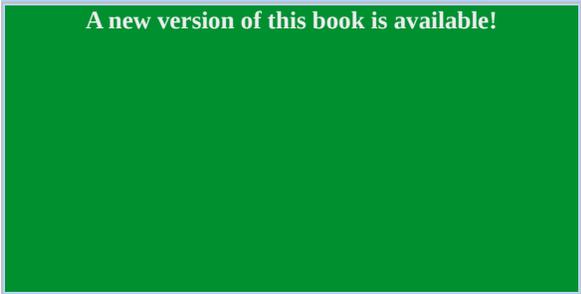
---

## **.wrapInner()**

Wrap an HTML structure around the content of each element in the set of matched elements.

POWERED BY **HERONOTE**

A new version of this book is available!



---

# Category: Class Attribute

These methods inspect and manipulate the CSS classes assigned to elements.

Also in: [Attributes](#) | [CSS](#)

---

## **.addClass()**

Adds the specified class(es) to each of the set of matched elements.

Also in: [Attributes](#) | [CSS](#)

---

## **.hasClass()**

Determine whether any of the matched elements are assigned the given class.

Also in: [Attributes](#) | [CSS](#)

---

## `.removeClass()`

Remove a single class, multiple classes, or all classes from each element in the set of matched elements.

Also in: [Attributes](#) | [CSS](#)

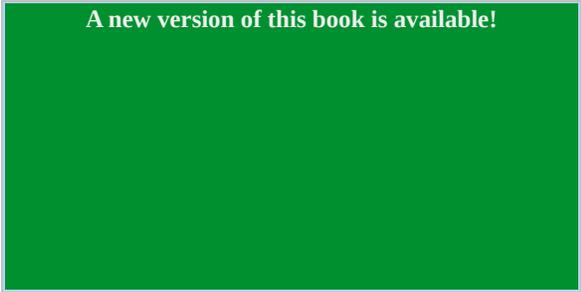
---

## **.toggleClass()**

Add or remove one or more classes from each element in the set of matched elements, depending on either the class's presence or the value of the switch argument.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

## Category: Copying

This method allows us to make copies of elements.

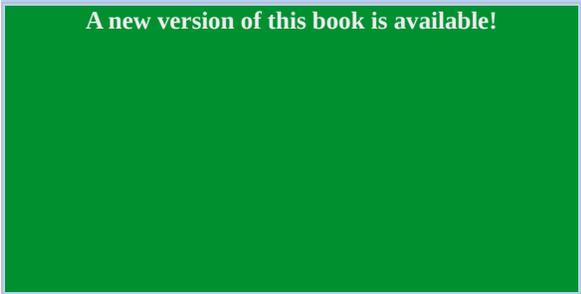
---

## `.clone()`

Create a deep copy of the set of matched elements.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

# Category: DOM Insertion, Around

These methods allow us to insert new content surrounding existing content.

Also in: [Manipulation](#) > [DOM Removal](#)

---

## **.unwrap()**

Remove the parents of the set of matched elements from the DOM, leaving the matched elements in their place.

---

## **.wrap()**

Wrap an HTML structure around each element in the set of matched elements.

---

## **.wrapAll()**

Wrap an HTML structure around all elements in the set of matched elements.

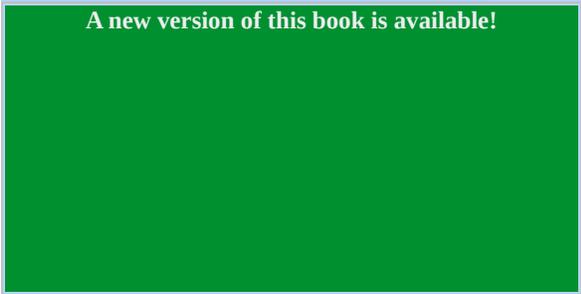
---

## **.wrapInner()**

Wrap an HTML structure around the content of each element in the set of matched elements.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

## Category: DOM Insertion, Inside

These methods allow us to insert new content inside an existing element.

---

## **.append()**

Insert content, specified by the parameter, to the end of each element in the set of matched elements.

---

## **.appendTo()**

Insert every element in the set of matched elements to the end of the target.

Also in: [Attributes](#)

---

## **.html()**

Get the HTML contents of the first element in the set of matched elements or set the HTML contents of every matched element.

---

## **.prepend()**

Insert content, specified by the parameter, to the beginning of each element in the set of matched elements.

---

## **.prependTo()**

Insert every element in the set of matched elements to the beginning of the target.

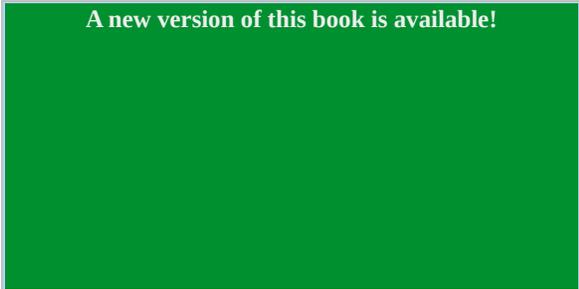
---

## **.text()**

Get the combined text contents of each element in the set of matched elements, including their descendants, or set the text contents of the matched elements.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

## Category: DOM Insertion, Outside

These methods allow us to insert new content outside an existing element.

---

## **.after()**

Insert content, specified by the parameter, after each element in the set of matched elements.

---

## **.before()**

Insert content, specified by the parameter, before each element in the set of matched elements.

---

## **.insertAfter()**

Insert every element in the set of matched elements after the target.

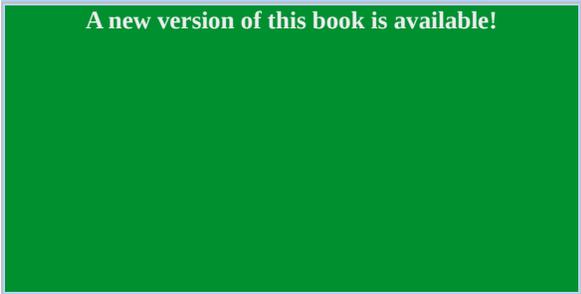
---

## **.insertBefore()**

Insert every element in the set of matched elements before the target.

POWERED BY **HERONOTE**

A new version of this book is available!



---

## Category: DOM Removal

These methods allow us to delete elements from the DOM.

---

## **.detach()**

Remove the set of matched elements from the DOM.

---

## **.empty()**

Remove all child nodes of the set of matched elements from the DOM.

---

## `.remove()`

Remove the set of matched elements from the DOM.

Also in: [Manipulation](#) > [DOM Insertion, Around](#)

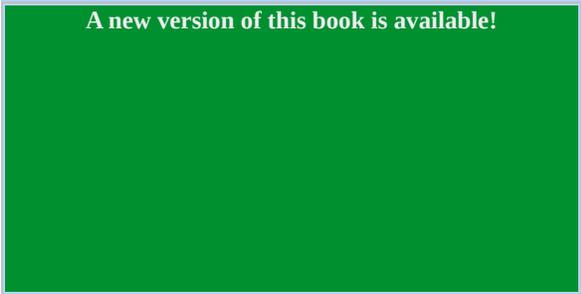
---

## **.unwrap()**

Remove the parents of the set of matched elements from the DOM, leaving the matched elements in their place.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

## Category: DOM Replacement

These methods are used to remove content from the DOM and replace it with new content.

---

## **.replaceAll()**

Replace each target element with the set of matched elements.

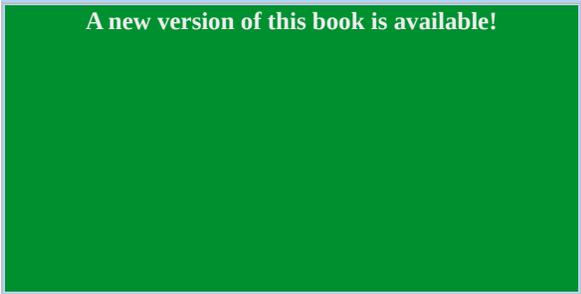
---

## **.replaceWith()**

Replace each element in the set of matched elements with the provided new content and return the set of elements that was removed.

POWERED BY **HERONOTE**

A new version of this book is available!



---

# Category: General Attributes

These methods get and set DOM attributes of elements

Also in: [Attributes](#)

---

## **.attr()**

Get the value of an attribute for the first element in the set of matched elements or set one or more attributes for every matched element.

Also in: [Attributes](#)

---

## **.prop()**

Get the value of a property for the first element in the set of matched elements or set one or more properties for every matched element.

Also in: [Attributes](#)

---

## **.removeAttr()**

Remove an attribute from each element in the set of matched elements.

Also in: [Attributes](#)

---

## **.removeProp()**

Remove a property for the set of matched elements.

Also in: [Attributes](#) | [Forms](#)

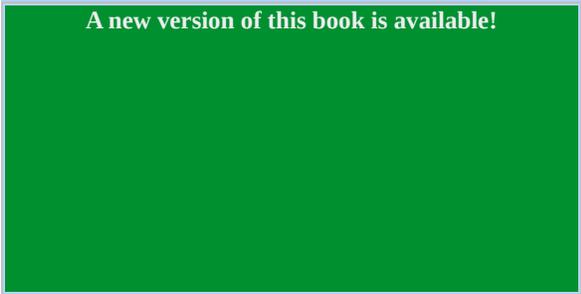
---

## `.val()`

Get the current value of the first element in the set of matched elements or set the value of every matched element.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

# Category: Style Properties

These methods get and set CSS-related properties of elements.

Also in: [CSS](#)

---

## `.css()`

Get the value of a style property for the first element in the set of matched elements or set one or more CSS properties for every matched element.

Also in: [CSS](#) | [Dimensions](#)

---

## **.height()**

Get the current computed height for the first element in the set of matched elements or set the height of every matched element.

Also in: [CSS](#) | [Dimensions](#)

---

## **.innerHeight()**

Get the current computed height for the first element in the set of matched elements, including padding but not border.

Also in: [CSS](#) | [Dimensions](#)

---

## **.innerWidth()**

Get the current computed width for the first element in the set of matched elements, including padding but not border.

Also in: [CSS](#) | [Offset](#)

---

## **.offset()**

Get the current coordinates of the first element, or set the coordinates of every element, in the set of matched elements, relative to the document.

Also in: [CSS](#) | [Dimensions](#)

---

## **.outerHeight()**

Get the current computed height for the first element in the set of matched elements, including padding, border, and optionally margin. Returns an integer (without "px") representation of the value or null if called on an empty set of elements.

Also in: [CSS](#) | [Dimensions](#)

---

## **.outerWidth()**

Get the current computed width for the first element in the set of matched elements, including padding and border.

Also in: [CSS](#) | [Offset](#)

---

## **.position()**

Get the current coordinates of the first element in the set of matched elements, relative to the offset parent.

Also in: [CSS](#) | [Offset](#)

---

## **.scrollLeft()**

Get the current horizontal position of the scroll bar for the first element in the set of matched elements or set the horizontal position of the scroll bar for every matched element.

Also in: [CSS](#) | [Offset](#)

---

## **.scrollTop()**

Get the current vertical position of the scroll bar for the first element in the set of matched elements or set the vertical position of the scroll bar for every matched element.

Also in: [CSS](#) | [Dimensions](#)

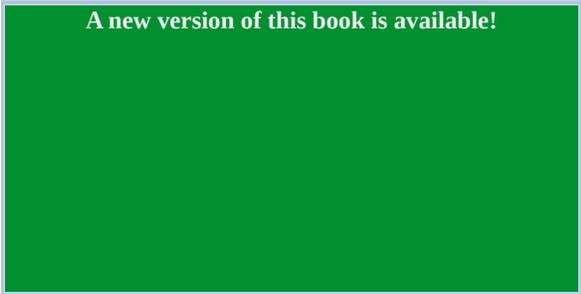
---

## `.width()`

Get the current computed width for the first element in the set of matched elements or set the width of every matched element.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

# Category: Miscellaneous

Also in: [Data](#) | [Miscellaneous](#) > [Data Storage](#)

---

## **.data()**

Store arbitrary data associated with the matched elements or return the value at the named data store for the first element in the set of matched elements.

Also in: [Miscellaneous](#) > [Collection Manipulation](#) | [Traversing](#)

---

## **.each()**

Iterate over a jQuery object, executing a function for each matched element.

Also in: [Miscellaneous](#) > [DOM Element Methods](#)

---

## **.get()**

Retrieve the DOM elements matched by the jQuery object.

Also in: [Miscellaneous](#) > [DOM Element Methods](#)

---

## `.index()`

Search for a given element from among the matched elements.

Also in: [Core](#) | [Miscellaneous](#) > [Setup Methods](#)

---

## jQuery.noConflict()

Relinquish jQuery's control of the \$ variable.

Also in: [Miscellaneous](#) > [Collection Manipulation](#) | [Forms](#) | [Ajax](#) > [Helper Functions](#)

---

## jQuery.param()

Create a serialized representation of an array or object, suitable for use in a URL query string or Ajax request.

Also in: [Data](#) | [Miscellaneous](#) > [Data Storage](#)

---

## **.removeData()**

Remove a previously-stored piece of data.

Also in: [Miscellaneous](#) > [DOM Element Methods](#)

---

## **.size()**

Return the number of elements in the jQuery object.

Also in: [Miscellaneous](#) > [DOM Element Methods](#)

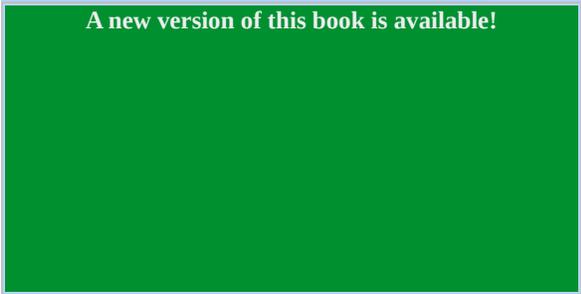
---

## **.toArray()**

Retrieve all the DOM elements contained in the jQuery set, as an array.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

# Category: Collection Manipulation

Also in: [Traversing](#)

---

## `.each()`

Iterate over a jQuery object, executing a function for each matched element.

Also in: [Forms](#) | [Ajax > Helper Functions](#)

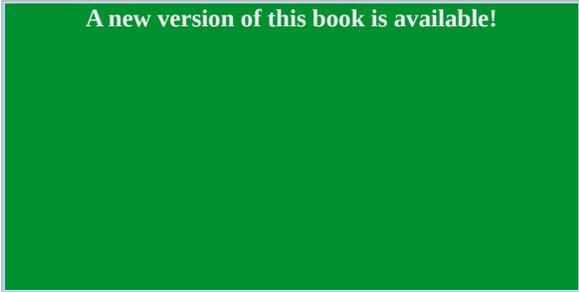
---

## jQuery.param()

Create a serialized representation of an array or object, suitable for use in a URL query string or Ajax request.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

# Category: Data Storage

These methods allow us to associate arbitrary data with specific DOM elements.

Also in: [Data](#)

---

## **.data()**

Store arbitrary data associated with the matched elements or return the value at the named data store for the first element in the set of matched elements.

Also in: [Data](#)

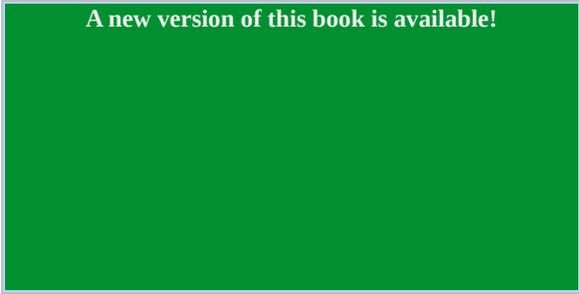
---

## **.removeData()**

Remove a previously-stored piece of data.

POWERED BY **HERONOTE**

A new version of this book is available!



---

## **Category: DOM Element Methods**

---

## **.get()**

Retrieve the DOM elements matched by the jQuery object.

---

## **.index()**

Search for a given element from among the matched elements.

---

## **.size()**

Return the number of elements in the jQuery object.

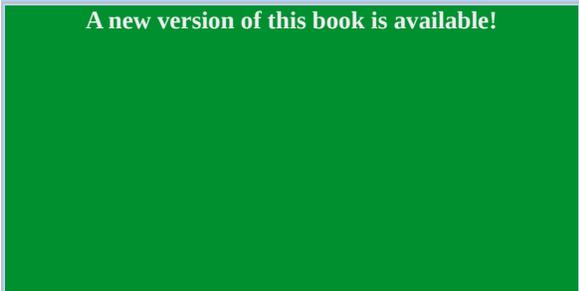
---

## **.toArray()**

Retrieve all the DOM elements contained in the jQuery set, as an array.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

# Category: Setup Methods

Also in: [Core](#)

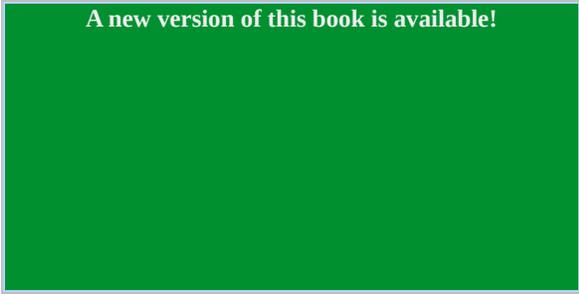
---

## jQuery.noConflict()

Relinquish jQuery's control of the \$ variable.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

# Category: Offset

Also in: [CSS](#) | [Manipulation](#) > [Style Properties](#)

---

## **.offset()**

Get the current coordinates of the first element, or set the coordinates of every element, in the set of matched elements, relative to the document.

Also in: [Traversing](#) > [Tree Traversal](#)

---

## `.offsetParent()`

Get the closest ancestor element that is positioned.

Also in: [CSS](#) | [Manipulation](#) > [Style Properties](#)

---

## **.position()**

Get the current coordinates of the first element in the set of matched elements, relative to the offset parent.

Also in: [CSS](#) | [Manipulation](#) > [Style Properties](#)

---

## **.scrollLeft()**

Get the current horizontal position of the scroll bar for the first element in the set of matched elements or set the horizontal position of the scroll bar for every matched element.

Also in: [CSS](#) | [Manipulation](#) > [Style Properties](#)

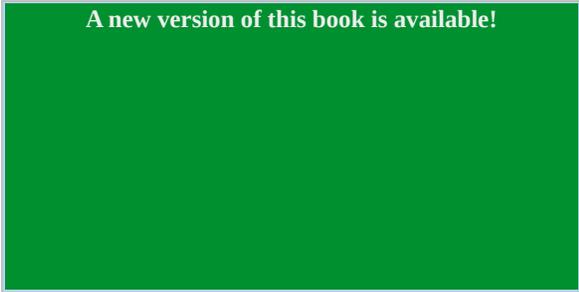
---

## **.scrollTop()**

Get the current vertical position of the scroll bar for the first element in the set of matched elements or set the vertical position of the scroll bar for every matched element.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

# Category: Properties

Also in: [Internals](#) | [Properties](#) > [Properties of jQuery Object Instances](#)

---

## **.context**

The DOM node context originally passed to `jQuery()`; if none was passed then context will likely be the document.

Also in: [Internals](#) | [Properties](#) > [Properties of jQuery Object Instances](#)

---

## .jquery

A string containing the jQuery version number.

Also in: [Properties](#) > [Properties of the Global jQuery Object](#) | [Utilities](#)

---

## jQuery.browser

Contains flags for the useragent, read from navigator.userAgent. We recommend against using this property; please try to use feature detection instead (see jQuery.support). jQuery.browser may be moved to a plugin in a future release of jQuery.

Also in: [Effects > Custom](#) | [Properties > Properties of the Global jQuery Object](#)

---

## jQuery.fx.interval

The rate (in milliseconds) at which animations fire.

Also in: [Effects > Custom](#) | [Properties > Properties of the Global jQuery Object](#)

---

## jQuery.fx.off

Globally disable all animations.

Also in: [Properties](#) > [Properties of the Global jQuery Object](#) | [Utilities](#)

---

## jQuery.support

A collection of properties that represent the presence of different browser features or bugs. Primarily intended for jQuery's internal use; specific properties may be removed when they are no longer needed internally to improve page startup performance.

Also in: [Properties](#) > [Properties of jQuery Object Instances](#)

---

## .length

The number of elements in the jQuery object.

Also in: [Internals](#) | [Properties](#) > [Properties of the Global jQuery Object](#)

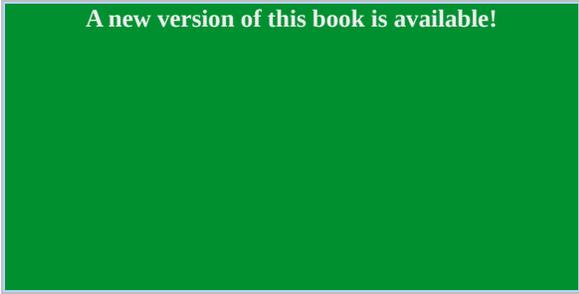
---

## **.selector**

A selector representing selector originally passed to jQuery().

POWERED BY **HERONOTE**

A new version of this book is available!



---

# Category: Properties of jQuery Object Instances

Each jQuery object created with the `jQuery()` function contains a number of properties alongside its methods. These properties allow us to inspect various attributes of the object.

Also in: [Internals](#)

---

## **.context**

The DOM node context originally passed to `jQuery()`; if none was passed then context will likely be the document.

Also in: [Internals](#)

---

## **.jquery**

A string containing the jQuery version number.

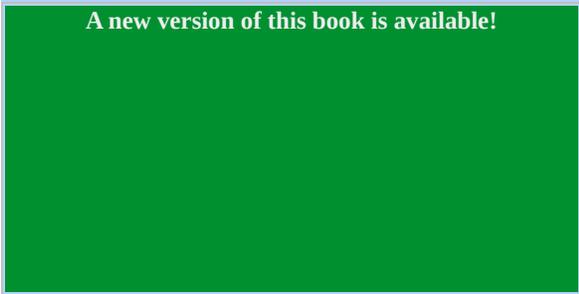
---

## **.length**

The number of elements in the jQuery object.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

# Category: Properties of the Global jQuery Object

These properties are associated with the global jQuery object.

Also in: [Utilities](#)

---

## jQuery.browser

Contains flags for the useragent, read from navigator.userAgent. We recommend against using this property; please try to use feature detection instead (see jQuery.support). jQuery.browser may be moved to a plugin in a future release of jQuery.

Also in: [Effects](#) > [Custom](#)

---

## jQuery.fx.interval

The rate (in milliseconds) at which animations fire.

Also in: [Effects](#) > [Custom](#)

---

## jQuery.fx.off

Globally disable all animations.

Also in: [Utilities](#)

---

## jQuery.support

A collection of properties that represent the presence of different browser features or bugs. Primarily intended for jQuery's internal use; specific properties may be removed when they are no longer needed internally to improve page startup performance.

Also in: [Internals](#)

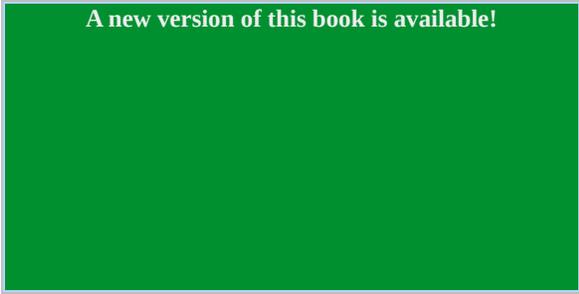
---

## **.selector**

A selector representing selector originally passed to jQuery().

POWERED BY **HERONOTE**

A new version of this book is available!



---

# Category: Selectors

Borrowing from CSS 1–3, and then adding its own, jQuery offers a powerful set of tools for matching a set of elements in a document.

To use any of the meta-characters ( such as `!"#$%&'()*+,-./:;<=>?@[\\]^_{|}~` ) as a literal part of a name, it must be escaped with with two backslashes: `\\`. For example, an element with `id="foo.bar"`, can use the selector `$("#foo\\.bar")`. The W3C CSS specification contains the [complete set of rules regarding valid CSS selectors](#). Also useful is the blog entry by Mathias Bynens on [CSS character escape sequences for identifiers](#).

Also in: [Selectors](#) > [Basic](#)

---

---

## All Selector (“\*”)

Selects all elements.

Also in: [Selectors > Basic Filter](#) | [Selectors > jQuery Extensions](#)

---

## **:animated Selector**

Select all elements that are in the progress of an animation at the time the selector is run.

Also in: [Selectors > Attribute](#)

---

## Attribute Contains Prefix Selector

### [name|="value"]

Selects elements that have the specified attribute with a value either equal to a given string or starting with that string followed by a hyphen (-).

Also in: [Selectors](#) > [Attribute](#)

---

## Attribute Contains Selector [name\*="value"]

Selects elements that have the specified attribute with a value containing the a given substring.

Also in: [Selectors](#) > [Attribute](#)

---

## Attribute Contains Word Selector

### `[name~="value"]`

Selects elements that have the specified attribute with a value containing a given word, delimited by spaces.

Also in: [Selectors](#) > [Attribute](#)

---

## Attribute Ends With Selector

### `[name$="value"]`

Selects elements that have the specified attribute with a value ending exactly with a given string. The comparison is case sensitive.

Also in: [Selectors](#) > [Attribute](#)

---

## Attribute Equals Selector [name="value"]

Selects elements that have the specified attribute with a value exactly equal to a certain value.

Also in: [Selectors > Attribute](#) | [Selectors > jQuery Extensions](#)

---

## Attribute Not Equal Selector

### [name!="value"]

Select elements that either don't have the specified attribute, or do have the specified attribute but not with a certain value.

Also in: [Selectors](#) > [Attribute](#)

---

## Attribute Starts With Selector

### `[name^="value"]`

Selects elements that have the specified attribute with a value beginning exactly with a given string.

Also in: [Selectors > Form](#) | [Selectors > jQuery Extensions](#)

---

## **:button Selector**

Selects all button elements and elements of type button.

Also in: [Selectors > Form](#) | [Selectors > jQuery Extensions](#)

---

## **:checkbox Selector**

Selects all elements of type checkbox.

Also in: [Selectors](#) > [Form](#)

---

## **:checked Selector**

Matches all elements that are checked.

Also in: [Selectors](#) > [Hierarchy](#)

---

## `child")" href="0D4EEA27BD.htm" rel=bookmark>Child Selector (“parent > child”)`

Selects all direct child elements specified by “child” of elements specified by “parent”.

Also in: [Selectors](#) > [Basic](#)

---

## Class Selector (“`.class`”)

Selects all elements with the given class.

Also in: [Selectors](#) > [Content Filter](#)

---

## **:contains() Selector**

Select all elements that contain the specified text.

Also in: [Selectors](#) > [Hierarchy](#)

---

## Descendant Selector (“ancestor descendant”)

Selects all elements that are descendants of a given ancestor.

Also in: [Selectors](#) > [Form](#)

---

---

## **:disabled Selector**

Selects all elements that are disabled.

Also in: [Selectors](#) > [Basic](#)

---

## Element Selector (“element”)

Selects all elements with the given tag name.

Also in: [Selectors](#) > [Content Filter](#)

---

## **:empty Selector**

Select all elements that have no children (including text nodes).

Also in: [Selectors](#) > [Form](#)

---

## **:enabled Selector**

Selects all elements that are enabled.

Also in: [Selectors > Basic Filter](#) | [Selectors > jQuery Extensions](#)

---

## **:eq() Selector**

Select the element at index n within the matched set.

Also in: [Selectors > Basic Filter](#) | [Selectors > jQuery Extensions](#)

---

## **:even Selector**

Selects even elements, zero-indexed. See also [odd](#).

Also in: [Selectors > Form](#) | [Selectors > jQuery Extensions](#)

---

## **:file Selector**

Selects all elements of type file.

Also in: [Selectors](#) > [Child Filter](#)

---

## **:first-child Selector**

Selects all elements that are the first child of their parent.

Also in: [Selectors](#) > [Child Filter](#)

---

## **:first-of-type Selector**

Selects all elements that are the first among siblings of the same element name.

Also in: [Selectors > Basic Filter](#) | [Selectors > jQuery Extensions](#)

---

## **:first Selector**

Selects the first matched element.

Also in: [Selectors > Basic Filter](#) | [Selectors > Form](#)

---

## **:focus Selector**

Selects element if it is currently focused.

Also in: [Selectors > Basic Filter](#) | [Selectors > jQuery Extensions](#)

---

## **:gt() Selector**

Select all elements at an index greater than index within the matched set.

Also in: [Selectors](#) > [Attribute](#)

---

## Has Attribute Selector [name]

Selects elements that have the specified attribute, with any value.

Also in: [Selectors > Content Filter](#) | [Selectors > jQuery Extensions](#)

---

## **:has() Selector**

Selects elements which contain at least one element that matches the specified selector.

Also in: [Selectors > Basic Filter](#) | [Selectors > jQuery Extensions](#)

---

## **:header Selector**

Selects all elements that are headers, like h1, h2, h3 and so on.

Also in: [Selectors > jQuery Extensions](#) | [Selectors > Visibility Filter](#)

---

## **:hidden Selector**

Selects all elements that are hidden.

Also in: [Selectors](#) > [Basic](#)

---

## ID Selector (“#id”)

Selects a single element with the given id attribute.

Also in: [Selectors > Form](#) | [Selectors > jQuery Extensions](#)

---

## **:image Selector**

Selects all elements of type image.

Also in: [Selectors > Form](#) | [Selectors > jQuery Extensions](#)

---

## **:input Selector**

Selects all input, textarea, select and button elements.

Also in: [Selectors](#) > [Basic Filter](#)

---

## **:lang Selector**

Selects all elements of the specified language.

Also in: [Selectors](#) > [Child Filter](#)

---

## **:last-child Selector**

Selects all elements that are the last child of their parent.

Also in: [Selectors](#) > [Child Filter](#)

---

## **:last-of-type Selector**

Selects all elements that are the last among siblings of the same element name.

Also in: [Selectors > Basic Filter](#) | [Selectors > jQuery Extensions](#)

---

## **:last Selector**

Selects the last matched element.

Also in: [Selectors > Basic Filter](#) | [Selectors > jQuery Extensions](#)

---

## **:lt()** Selector

Select all elements at an index less than index within the matched set.

Also in: [Selectors](#) > [Attribute](#)

---

## Multiple Attribute Selector `[name="value"]` `[name2="value2"]`

Matches elements that match all of the specified attribute filters.

Also in: [Selectors](#) > [Basic](#)

---

---

## Multiple Selector (“selector1, selector2, selectorN”)

Selects the combined results of all the specified selectors.

Also in: [Selectors](#) > [Hierarchy](#)

---

---

## Next Adjacent Selector (“prev + next”)

Selects all next elements matching “next” that are immediately preceded by a sibling “prev”.

Also in: [Selectors > Hierarchy](#)

---

## Next Siblings Selector (“prev ~ siblings”)

Selects all sibling elements that follow after the “prev” element, have the same parent, and match the filtering “siblings” selector.

Also in: [Selectors](#) > [Basic Filter](#)

---

## **:not() Selector**

Selects all elements that do not match the given selector.

Also in: [Selectors](#) > [Child Filter](#)

---

## **:nth-child() Selector**

Selects all elements that are the nth-child of their parent.

Also in: [Selectors](#) > [Child Filter](#)

---

## **:nth-last-child() Selector**

Selects all elements that are the nth-child of their parent, counting from the last element to the first.

Also in: [Selectors](#) > [Child Filter](#)

---

## **:nth-last-of-type()** Selector

Selects all elements that are the nth-child of their parent, counting from the last element to the first.

Also in: [Selectors](#) > [Child Filter](#)

---

## **:nth-of-type() Selector**

Selects all elements that are the nth child of their parent in relation to siblings with the same element name.

Also in: [Selectors > Basic Filter](#) | [Selectors > jQuery Extensions](#)

---

## **:odd Selector**

Selects odd elements, zero-indexed. See also even.

Also in: [Selectors](#) > [Child Filter](#)

---

## **:only-child Selector**

Selects all elements that are the only child of their parent.

Also in: [Selectors](#) > [Child Filter](#)

---

## **:only-of-type Selector**

Selects all elements that have no siblings with the same element name.

Also in: [Selectors > Content Filter](#) | [Selectors > jQuery Extensions](#)

---

## **:parent Selector**

Select all elements that that have at least one child node (either an element or text).

Also in: [Selectors > Form](#) | [Selectors > jQuery Extensions](#)

---

## **:password Selector**

Selects all elements of type password.

Also in: [Selectors > Form](#) | [Selectors > jQuery Extensions](#)

---

## **:radio Selector**

Selects all elements of type radio.

Also in: [Selectors > Form](#) | [Selectors > jQuery Extensions](#)

---

## **:reset Selector**

Selects all elements of type reset.

Also in: [Selectors](#) > [Basic Filter](#)

---

## **:root Selector**

Selects the element that is the root of the document.

Also in: [Selectors > Form](#) | [Selectors > jQuery Extensions](#)

---

## **:selected Selector**

Selects all elements that are selected.

Also in: [Selectors > Form](#) | [Selectors > jQuery Extensions](#)

---

## **:submit Selector**

Selects all elements of type submit.

Also in: [Selectors](#) > [Basic Filter](#)

---

## **:target Selector**

Selects the target element indicated by the fragment identifier of the document's URI.

Also in: [Selectors > Form](#) | [Selectors > jQuery Extensions](#)

---

## **:text Selector**

Selects all elements of type text.

Also in: [Selectors > jQuery Extensions](#) | [Selectors > Visibility Filter](#)

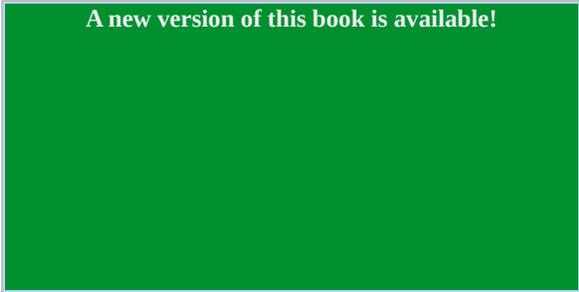
---

## **:visible Selector**

Selects all elements that are visible.

POWERED BY **HERONOTE**

A new version of this book is available!



# Category: Attribute

The CSS specification allows elements to be identified by their attributes. While not supported by some older browsers for the purpose of styling documents, jQuery allows you to employ them regardless of the browser being used.

When using any of the following attribute selectors, you should account for attributes that have multiple, space-separated values. Since these selectors see attribute values as a single string, this selector, for example, `$("a[rel='nofollow']")`, will select `<a href="example.html" rel="nofollow">Some text</a>` but not `<a href="example.html" rel="nofollow foe">Some text</a>`.

Attribute values in selector expressions **must** follow the rules for W3C CSS selectors; in general, that means anything other than a [valid identifier](#) should be surrounded by quotation marks.

double quotes inside single quotes: `$('a[rel="nofollow self"]')`

single quotes inside double quotes: `$("a[rel='nofollow self']")`

escaped single quotes inside single quotes: `$('a[rel=\'nofollow self\']')`

escaped double quotes inside double quotes: `$("a[rel=\'"nofollow self"]")`

The variation you choose is generally a matter of style or convenience.

**Note:** In jQuery 1.3 `[@attr]` style selectors were removed (they were previously deprecated in jQuery 1.2). Simply remove the “@” symbol from your selectors in order to make them work again.

---

## Attribute Contains Prefix Selector

**[name|"value"]**

Selects elements that have the specified attribute with a value either equal to a given string or starting with that string followed by a hyphen (-).

---

## Attribute Contains Selector [name\*="value"]

Selects elements that have the specified attribute with a value containing the a given substring.

---

## Attribute Contains Word Selector

**[name~="value"]**

Selects elements that have the specified attribute with a value containing a given word, delimited by spaces.

---

## Attribute Ends With Selector

**[name\$="value"]**

Selects elements that have the specified attribute with a value ending exactly with a given string. The comparison is case sensitive.

---

## Attribute Equals Selector [name="value"]

Selects elements that have the specified attribute with a value exactly equal to a certain value.

Also in: [Selectors](#) > [jQuery Extensions](#)

---

## Attribute Not Equal Selector

**[name!="value"]**

Select elements that either don't have the specified attribute, or do have the specified attribute but not with a certain value.

---

## Attribute Starts With Selector

**[name^="value"]**

Selects elements that have the specified attribute with a value beginning exactly with a given string.

---

## Has Attribute Selector [name]

Selects elements that have the specified attribute, with any value.

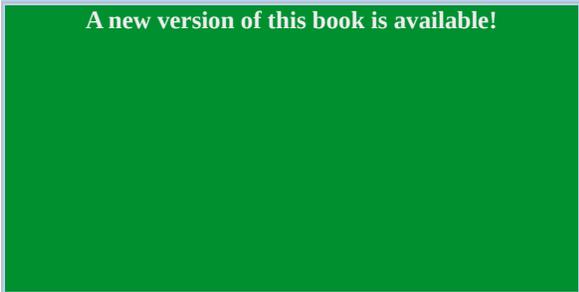
---

## Multiple Attribute Selector [name="value"] [name2="value2"]

Matches elements that match all of the specified attribute filters.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

## Category: Basic

The following selectors are based on the Cascading Style Sheet 1 specification, as outlined by the W3C. For more information about the specifications, visit <http://www.w3.org/Style/CSS/#specs>.

---

## All Selector (“\*”)

Selects all elements.

---

## **Class Selector (“`.class`”)**

Selects all elements with the given class.

---

## Element Selector (“element”)

Selects all elements with the given tag name.

---

## **ID Selector (“#id”)**

Selects a single element with the given id attribute.

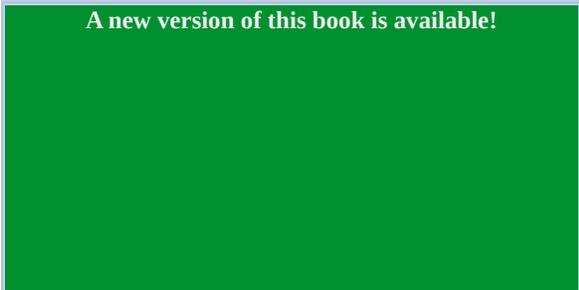
---

## Multiple Selector (“selector1, selector2, selectorN”)

Selects the combined results of all the specified selectors.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

# Category: Basic Filter

Also in: [Selectors](#) > [jQuery Extensions](#)

---

## **:animated Selector**

Select all elements that are in the progress of an animation at the time the selector is run.

Also in: [Selectors](#) > [jQuery Extensions](#)

---

## **:eq() Selector**

Select the element at index n within the matched set.

Also in: [Selectors](#) > [jQuery Extensions](#)

---

## **:even Selector**

Selects even elements, zero-indexed. See also [odd](#).

Also in: [Selectors](#) > [jQuery Extensions](#)

---

## **:first Selector**

Selects the first matched element.

Also in: [Selectors](#) > [Form](#)

---

## **:focus Selector**

Selects element if it is currently focused.

Also in: [Selectors](#) > [jQuery Extensions](#)

---

## **:gt() Selector**

Select all elements at an index greater than index within the matched set.

Also in: [Selectors > jQuery Extensions](#)

---

## **:header Selector**

Selects all elements that are headers, like h1, h2, h3 and so on.

---

## **:lang Selector**

Selects all elements of the specified language.

Also in: [Selectors](#) > [jQuery Extensions](#)

---

## **:last Selector**

Selects the last matched element.

Also in: [Selectors](#) > [jQuery Extensions](#)

---

## **:lt() Selector**

Select all elements at an index less than index within the matched set.

---

## **:not() Selector**

Selects all elements that do not match the given selector.

Also in: [Selectors](#) > [jQuery Extensions](#)

---

## **:odd Selector**

Selects odd elements, zero-indexed. See also even.

---

## **:root Selector**

Selects the element that is the root of the document.

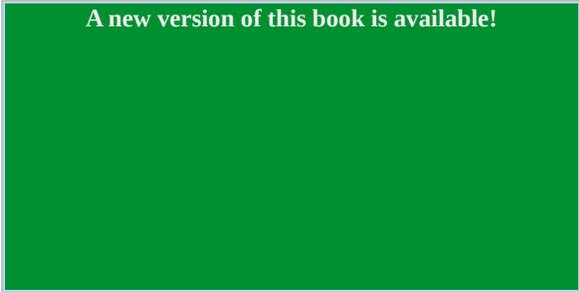
---

## **:target Selector**

Selects the target element indicated by the fragment identifier of the document's URI.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

## Category: Child Filter

---

## **:first-child Selector**

Selects all elements that are the first child of their parent.

---

## **:first-of-type Selector**

Selects all elements that are the first among siblings of the same element name.

---

## **:last-child Selector**

Selects all elements that are the last child of their parent.

---

## **:last-of-type Selector**

Selects all elements that are the last among siblings of the same element name.

---

## **:nth-child() Selector**

Selects all elements that are the nth-child of their parent.

---

## **:nth-last-child()** Selector

Selects all elements that are the nth-child of their parent, counting from the last element to the first.

---

## **:nth-last-of-type() Selector**

Selects all elements that are the nth-child of their parent, counting from the last element to the first.

---

## **:nth-of-type() Selector**

Selects all elements that are the nth child of their parent in relation to siblings with the same element name.

---

## **:only-child Selector**

Selects all elements that are the only child of their parent.

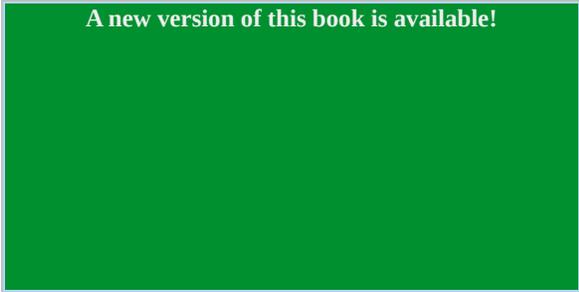
---

## :only-of-type Selector

Selects all elements that have no siblings with the same element name.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

## **Category: Content Filter**

---

## **:contains() Selector**

Select all elements that contain the specified text.

---

## **:empty Selector**

Select all elements that have no children (including text nodes).

Also in: [Selectors](#) > [jQuery Extensions](#)

---

## **:has() Selector**

Selects elements which contain at least one element that matches the specified selector.

Also in: [Selectors > jQuery Extensions](#)

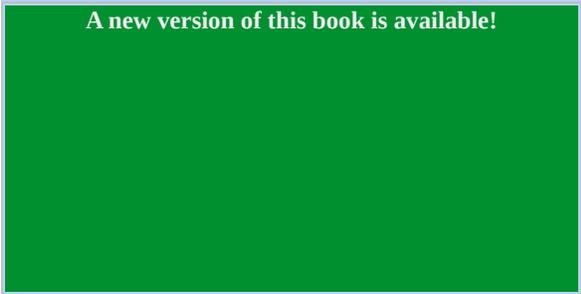
---

## **:parent Selector**

Select all elements that that have at least one child node (either an element or text).

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

# Category: Form

Also in: [Selectors](#) > [jQuery Extensions](#)

---

## **:button Selector**

Selects all button elements and elements of type button.

Also in: [Selectors](#) > [jQuery Extensions](#)

---

## **:checkbox Selector**

Selects all elements of type checkbox.

---

## **:checked Selector**

Matches all elements that are checked.

---

## **:disabled Selector**

Selects all elements that are disabled.

---

## **:enabled Selector**

Selects all elements that are enabled.

Also in: [Selectors](#) > [jQuery Extensions](#)

---

## **:file Selector**

Selects all elements of type file.

Also in: [Selectors](#) > [Basic Filter](#)

---

## **:focus Selector**

Selects element if it is currently focused.

Also in: [Selectors](#) > [jQuery Extensions](#)

---

## **:image Selector**

Selects all elements of type image.

Also in: [Selectors](#) > [jQuery Extensions](#)

---

## **:input Selector**

Selects all input, textarea, select and button elements.

Also in: [Selectors](#) > [jQuery Extensions](#)

---

## **:password Selector**

Selects all elements of type password.

Also in: [Selectors](#) > [jQuery Extensions](#)

---

## **:radio Selector**

Selects all elements of type radio.

Also in: [Selectors](#) > [jQuery Extensions](#)

---

## **:reset Selector**

Selects all elements of type reset.

Also in: [Selectors](#) > [jQuery Extensions](#)

---

## **:selected Selector**

Selects all elements that are selected.

Also in: [Selectors](#) > [jQuery Extensions](#)

---

## **:submit Selector**

Selects all elements of type submit.

Also in: [Selectors](#) > [jQuery Extensions](#)

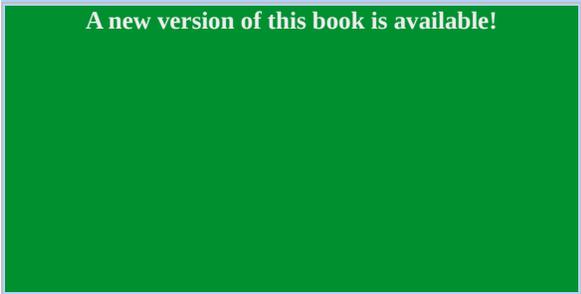
---

## **:text Selector**

Selects all elements of type text.

POWERED BY **HERONOTE**

A new version of this book is available!



---

## Category: Hierarchy

---

**child")" href="0D4EEA27BD.htm"  
rel="bookmark">Child Selector (“parent >  
child”)**

Selects all direct child elements specified by “child” of elements specified by “parent”.

---

## Descendant Selector (“ancestor descendant”)

Selects all elements that are descendants of a given ancestor.

---

---

## Next Adjacent Selector (“prev + next”)

Selects all next elements matching “next” that are immediately preceded by a sibling “prev”.

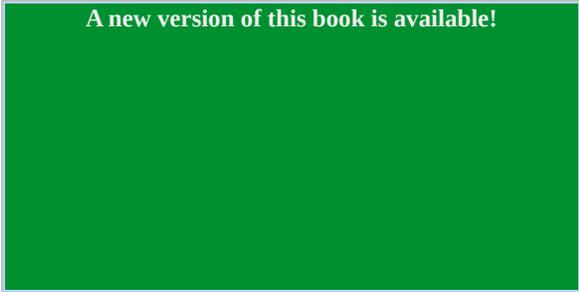
---

## Next Siblings Selector (“prev ~ siblings”)

Selects all sibling elements that follow after the “prev” element, have the same parent, and match the filtering “siblings” selector.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

# Category: jQuery Extensions

jQuery has extended the CSS3 selectors with the following selectors. Because these selectors are jQuery extension and not part of the CSS specification, queries using them cannot take advantage of the performance boost provided by the native DOM `querySelectorAll()` method. To achieve the best performance when using these selectors, first select some elements using a pure CSS selector, then use `.filter()`.

Also in: [Selectors](#) > [Basic Filter](#)

---

## **:animated Selector**

Select all elements that are in the progress of an animation at the time the selector is run.

Also in: [Selectors](#) > [Attribute](#)

---

## Attribute Not Equal Selector

### [name!="value"]

Select elements that either don't have the specified attribute, or do have the specified attribute but not with a certain value.

Also in: [Selectors](#) > [Form](#)

---

## **:button Selector**

Selects all button elements and elements of type button.

Also in: [Selectors](#) > [Form](#)

---

## **:checkbox Selector**

Selects all elements of type checkbox.

Also in: [Selectors](#) > [Basic Filter](#)

---

## **:eq() Selector**

Select the element at index n within the matched set.

Also in: [Selectors](#) > [Basic Filter](#)

---

## **:even Selector**

Selects even elements, zero-indexed. See also [odd](#).

Also in: [Selectors](#) > [Form](#)

---

## **:file Selector**

Selects all elements of type file.

Also in: [Selectors](#) > [Basic Filter](#)

---

## **:first Selector**

Selects the first matched element.

Also in: [Selectors](#) > [Basic Filter](#)

---

## **:gt() Selector**

Select all elements at an index greater than index within the matched set.

Also in: [Selectors > Content Filter](#)

---

## **:has() Selector**

Selects elements which contain at least one element that matches the specified selector.

Also in: [Selectors](#) > [Basic Filter](#)

---

## **:header Selector**

Selects all elements that are headers, like h1, h2, h3 and so on.

Also in: [Selectors](#) > [Visibility Filter](#)

---

## **:hidden Selector**

Selects all elements that are hidden.

Also in: [Selectors](#) > [Form](#)

---

## **:image Selector**

Selects all elements of type image.

Also in: [Selectors](#) > [Form](#)

---

## **:input Selector**

Selects all input, textarea, select and button elements.

Also in: [Selectors](#) > [Basic Filter](#)

---

## **:last Selector**

Selects the last matched element.

Also in: [Selectors](#) > [Basic Filter](#)

---

## **:lt() Selector**

Select all elements at an index less than index within the matched set.

Also in: [Selectors](#) > [Basic Filter](#)

---

## **:odd Selector**

Selects odd elements, zero-indexed. See also even.

Also in: [Selectors](#) > [Content Filter](#)

---

## **:parent Selector**

Select all elements that that have at least one child node (either an element or text).

Also in: [Selectors > Form](#)

---

## **:password Selector**

Selects all elements of type password.

Also in: [Selectors](#) > [Form](#)

---

## **:radio Selector**

Selects all elements of type radio.

Also in: [Selectors](#) > [Form](#)

---

## **:reset Selector**

Selects all elements of type reset.

Also in: [Selectors](#) > [Form](#)

---

## **:selected Selector**

Selects all elements that are selected.

Also in: [Selectors](#) > [Form](#)

---

## **:submit Selector**

Selects all elements of type submit.

Also in: [Selectors](#) > [Form](#)

---

## **:text Selector**

Selects all elements of type text.

Also in: [Selectors](#) > [Visibility Filter](#)

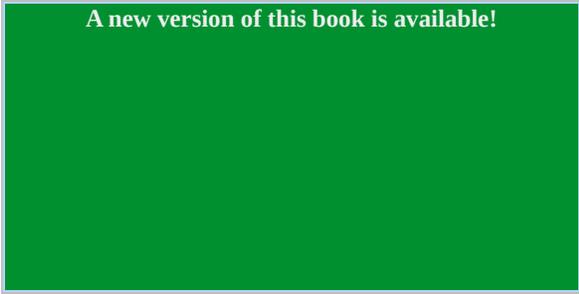
---

## **:visible Selector**

Selects all elements that are visible.

POWERED BY **HERONOTE**

A new version of this book is available!



---

# Category: Visibility Filter

Also in: [Selectors](#) > [jQuery Extensions](#)

---

## **:hidden Selector**

Selects all elements that are hidden.

Also in: [Selectors](#) > [jQuery Extensions](#)

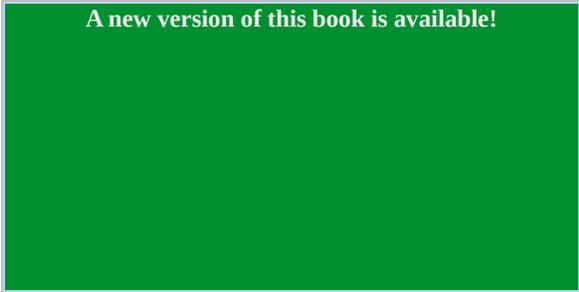
---

## **:visible Selector**

Selects all elements that are visible.

POWERED BY **HERONOTE**

A new version of this book is available!



---

# Category: Traversing

Also in: [Traversing](#) > [Miscellaneous Traversing](#)

---

## **.add()**

Add elements to the set of matched elements.

Also in: [Traversing](#) > [Miscellaneous Traversing](#)

---

## **.addBack()**

Add the previous set of elements on the stack to the current set, optionally filtered by a selector.

Also in: [Traversing](#) > [Miscellaneous Traversing](#)

---

## **.andSelf()**

Add the previous set of elements on the stack to the current set.

Also in: [Traversing](#) > [Tree Traversal](#)

---

## **.children()**

Get the children of each element in the set of matched elements, optionally filtered by a selector.

Also in: [Traversing > Tree Traversal](#)

---

## `.closest()`

For each element in the set, get the first element that matches the selector by testing the element itself and traversing up through its ancestors in the DOM tree.

Also in: [Traversing](#) > [Miscellaneous Traversing](#)

---

## **.contents()**

Get the children of each element in the set of matched elements, including text and comment nodes.

Also in: [Miscellaneous](#) > [Collection Manipulation](#)

---

## `.each()`

Iterate over a jQuery object, executing a function for each matched element.

Also in: [Traversing](#) > [Miscellaneous Traversing](#)

---

## **.end()**

End the most recent filtering operation in the current chain and return the set of matched elements to its previous state.

Also in: [Traversing](#) > [Filtering](#)

---

## `.eq()`

Reduce the set of matched elements to the one at the specified index.

Also in: [Traversing](#) > [Filtering](#)

---

## **.filter()**

Reduce the set of matched elements to those that match the selector or pass the function's test.

Also in: [Traversing > Tree Traversal](#)

---

## **.find()**

Get the descendants of each element in the current set of matched elements, filtered by a selector, jQuery object, or element.

Also in: [Traversing](#) > [Filtering](#)

---

## **.first()**

Reduce the set of matched elements to the first in the set.

Also in: [Traversing](#) > [Filtering](#)

---

## `.has()`

Reduce the set of matched elements to those that have a descendant that matches the selector or DOM element.

Also in: [Traversing](#) > [Filtering](#)

---

## `.is()`

Check the current matched set of elements against a selector, element, or jQuery object and return true if at least one of these elements matches the given arguments.

Also in: [Traversing](#) > [Filtering](#)

---

## **.last()**

Reduce the set of matched elements to the final one in the set.

Also in: [Traversing](#) > [Filtering](#)

---

## `.map()`

Pass each element in the current matched set through a function, producing a new jQuery object containing the return values.

Also in: [Traversing > Tree Traversal](#)

---

## `.next()`

Get the immediately following sibling of each element in the set of matched elements. If a selector is provided, it retrieves the next sibling only if it matches that selector.

Also in: [Traversing](#) > [Tree Traversal](#)

---

## **.nextAll()**

Get all following siblings of each element in the set of matched elements, optionally filtered by a selector.

Also in: [Traversing > Tree Traversal](#)

---

## **.nextUntil()**

Get all following siblings of each element up to but not including the element matched by the selector, DOM node, or jQuery object passed.

Also in: [Traversing > Filtering](#) | [Traversing > Miscellaneous Traversing](#)

---

## **.not()**

Remove elements from the set of matched elements.

Also in: [Offset](#) | [Traversing](#) > [Tree Traversal](#)

---

## **.offsetParent()**

Get the closest ancestor element that is positioned.

Also in: [Traversing](#) > [Tree Traversal](#)

---

## **.parent()**

Get the parent of each element in the current set of matched elements, optionally filtered by a selector.

Also in: [Traversing > Tree Traversal](#)

---

## **.parents()**

Get the ancestors of each element in the current set of matched elements, optionally filtered by a selector.

Also in: [Traversing](#) > [Tree Traversal](#)

---

## **.parentsUntil()**

Get the ancestors of each element in the current set of matched elements, up to but not including the element matched by the selector, DOM node, or jQuery object.

Also in: [Traversing](#) > [Tree Traversal](#)

---

## `.prev()`

Get the immediately preceding sibling of each element in the set of matched elements, optionally filtered by a selector.

Also in: [Traversing > Tree Traversal](#)

---

## **.prevAll()**

Get all preceding siblings of each element in the set of matched elements, optionally filtered by a selector.

Also in: [Traversing > Tree Traversal](#)

---

## **.prevUntil()**

Get all preceding siblings of each element up to but not including the element matched by the selector, DOM node, or jQuery object.

Also in: [Traversing > Tree Traversal](#)

---

## **.siblings()**

Get the siblings of each element in the set of matched elements, optionally filtered by a selector.

Also in: [Traversing](#) > [Filtering](#)

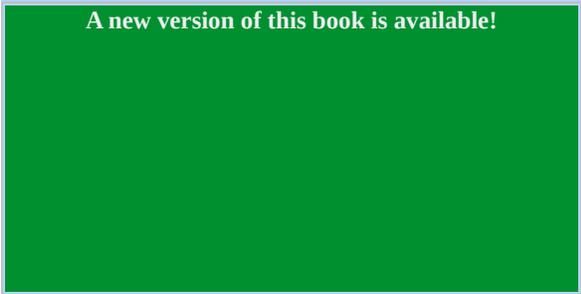
---

## **.slice()**

Reduce the set of matched elements to a subset specified by a range of indices.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

# Category: Filtering

---

**.eq()**

Reduce the set of matched elements to the one at the specified index.

---

## **.filter()**

Reduce the set of matched elements to those that match the selector or pass the function's test.

---

## **.first()**

Reduce the set of matched elements to the first in the set.

---

## **.has()**

Reduce the set of matched elements to those that have a descendant that matches the selector or DOM element.

---

## **.is()**

Check the current matched set of elements against a selector, element, or jQuery object and return true if at least one of these elements matches the given arguments.

---

## **.last()**

Reduce the set of matched elements to the final one in the set.

---

## `.map()`

Pass each element in the current matched set through a function, producing a new jQuery object containing the return values.

Also in: [Traversing](#) > [Miscellaneous Traversing](#)

---

## **.not()**

Remove elements from the set of matched elements.

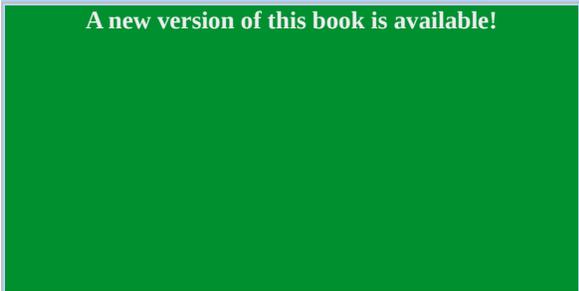
---

## **.slice()**

Reduce the set of matched elements to a subset specified by a range of indices.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

# Category: Miscellaneous Traversing

---

## **.add()**

Add elements to the set of matched elements.

---

## **.addBack()**

Add the previous set of elements on the stack to the current set, optionally filtered by a selector.

---

## **.andSelf()**

Add the previous set of elements on the stack to the current set.

---

## **.contents()**

Get the children of each element in the set of matched elements, including text and comment nodes.

---

## **.end()**

End the most recent filtering operation in the current chain and return the set of matched elements to its previous state.

Also in: [Traversing](#) > [Filtering](#)

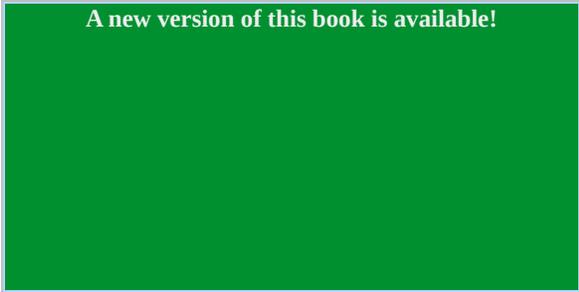
---

**.not()**

Remove elements from the set of matched elements.

POWERED BY **HERONOTE**

A new version of this book is available!



---

# Category: Tree Traversal

---

## **.children()**

Get the children of each element in the set of matched elements, optionally filtered by a selector.

---

## **.closest()**

For each element in the set, get the first element that matches the selector by testing the element itself and traversing up through its ancestors in the DOM tree.

---

## **.find()**

Get the descendants of each element in the current set of matched elements, filtered by a selector, jQuery object, or element.

---

## **.next()**

Get the immediately following sibling of each element in the set of matched elements. If a selector is provided, it retrieves the next sibling only if it matches that selector.

---

## **.nextAll()**

Get all following siblings of each element in the set of matched elements, optionally filtered by a selector.

---

## **.nextUntil()**

Get all following siblings of each element up to but not including the element matched by the selector, DOM node, or jQuery object passed.

Also in: [Offset](#)

---

## **.offsetParent()**

Get the closest ancestor element that is positioned.

---

## **.parent()**

Get the parent of each element in the current set of matched elements, optionally filtered by a selector.

---

## **.parents()**

Get the ancestors of each element in the current set of matched elements, optionally filtered by a selector.

---

## **.parentsUntil()**

Get the ancestors of each element in the current set of matched elements, up to but not including the element matched by the selector, DOM node, or jQuery object.

---

## **.prev()**

Get the immediately preceding sibling of each element in the set of matched elements, optionally filtered by a selector.

---

## **.prevAll()**

Get all preceding siblings of each element in the set of matched elements, optionally filtered by a selector.

---

## **.prevUntil()**

Get all preceding siblings of each element up to but not including the element matched by the selector, DOM node, or jQuery object.

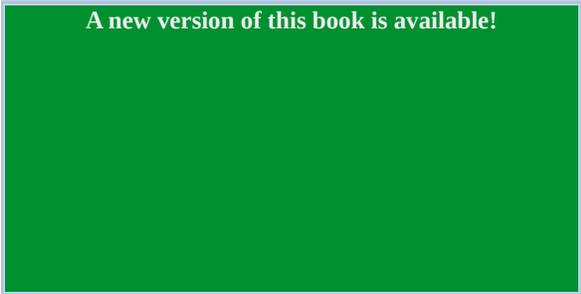
---

## **.siblings()**

Get the siblings of each element in the set of matched elements, optionally filtered by a selector.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

# Category: Utilities

Also in: [Effects](#) > [Custom](#) | [Data](#)

---

## **.clearQueue()**

Remove from the queue all items that have not yet been run.

Also in: [Effects](#) > [Custom](#) | [Data](#)

---

## **.dequeue()**

Execute the next function on the queue for the matched elements.

---

## jQuery.boxModel

Deprecated in jQuery 1.3 (see [jQuery.support](#)). States if the current page, in the user's browser, is being rendered using the W3C CSS Box Model.

Also in: [Properties](#) > [Properties of the Global jQuery Object](#)

---

## jQuery.browser

Contains flags for the useragent, read from navigator.userAgent. We recommend against using this property; please try to use feature detection instead (see [jQuery.support](#)). jQuery.browser may be moved to a plugin in a future release of jQuery.

---

## jQuery.contains()

Check to see if a DOM element is a descendant of another DOM element.

Also in: [Data](#)

---

## jQuery.data()

Store arbitrary data associated with the specified element and/or return the value that was set.

Also in: [Data](#)

---

## jQuery.dequeue()

Execute the next function on the queue for the matched element.

---

## jQuery.each()

A generic iterator function, which can be used to seamlessly iterate over both objects and arrays. Arrays and array-like objects with a length property (such as a function's arguments object) are iterated by numeric index, from 0 to length-1. Other objects are iterated via their named properties.

---

## jQuery.extend()

Merge the contents of two or more objects together into the first object.

---

## jQuery.globalEval()

Execute some JavaScript code globally.

---

## jQuery.grep()

Finds the elements of an array which satisfy a filter function. The original array is not affected.

---

## jQuery.inArray()

Search for a specified value within an array and return its index (or -1 if not found).

---

## jQuery.isArray()

Determine whether the argument is an array.

---

## jQuery.isEmptyObject()

Check to see if an object is empty (contains no enumerable properties).

---

## jQuery.isFunction()

Determine if the argument passed is a Javascript function object.

---

## jQuery.isNumeric()

Determines whether its argument is a number.

---

## jQuery.isPlainObject()

Check to see if an object is a plain object (created using “{}” or “new Object”).

---

## jQuery.isWindow()

Determine whether the argument is a window.

---

## jQuery.isXMLDoc()

Check to see if a DOM node is within an XML document (or is an XML document).

---

## jQuery.makeArray()

Convert an array-like object into a true JavaScript array.

---

## jQuery.map()

Translate all items in an array or object to new array of items.

---

## jQuery.merge()

Merge the contents of two arrays together into the first array.

---

## jQuery.noop()

An empty function.

---

## jQuery.now()

Return a number representing the current time.

---

## jQuery.parseHTML()

Parses a string into an array of DOM nodes.

---

## jQuery.parseJSON()

Takes a well-formed JSON string and returns the resulting JavaScript object.

---

## jQuery.parseXML()

Parses a string into an XML document.

Also in: [Events](#) > [Event Handler Attachment](#)

---

## jQuery.proxy()

Takes a function and returns a new one that will always have a particular context.

Also in: [Data](#)

---

## jQuery.queue()

Show or manipulate the queue of functions to be executed on the matched element.

Also in: [Data](#)

---

## jQuery.removeData()

Remove a previously-stored piece of data.

Also in: [Properties](#) > [Properties of the Global jQuery Object](#)

---

## jQuery.support

A collection of properties that represent the presence of different browser features or bugs. Primarily intended for jQuery's internal use; specific properties may be removed when they are no longer needed internally to improve page startup performance.

---

## jQuery.trim()

Remove the whitespace from the beginning and end of a string.

---

## jQuery.type()

Determine the internal JavaScript `[[Class]]` of an object.

---

## jQuery.unique()

Sorts an array of DOM elements, in place, with the duplicates removed. Note that this only works on arrays of DOM elements, not strings or numbers.

Also in: [Effects](#) > [Custom](#) | [Data](#)

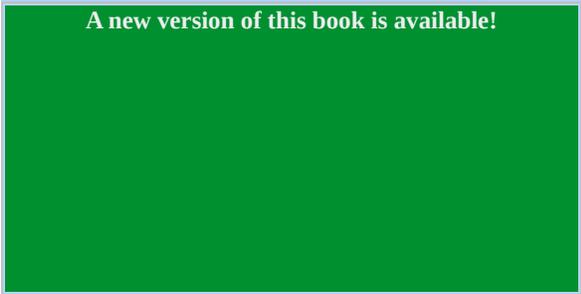
---

## `.queue()`

Show or manipulate the queue of functions to be executed on the matched elements.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

# Category: Version

Also in: [Traversing](#) > [Miscellaneous Traversing](#)

---

## **.add()**

Add elements to the set of matched elements.

Also in: [Traversing](#) > [Miscellaneous Traversing](#)

---

## **.addBack()**

Add the previous set of elements on the stack to the current set, optionally filtered by a selector.

Also in: [Attributes](#) | [Manipulation > Class Attribute](#) | [CSS](#)

---

## **.addClass()**

Adds the specified class(es) to each of the set of matched elements.

Also in: [Manipulation > DOM Insertion, Outside](#)

---

## **.after()**

Insert content, specified by the parameter, after each element in the set of matched elements.

Also in: [Ajax > Global Ajax Event Handlers](#)

---

## **.ajaxComplete()**

Register a handler to be called when Ajax requests complete. This is an AjaxEvent.

Also in: [Ajax > Global Ajax Event Handlers](#)

---

## **.ajaxError()**

Register a handler to be called when Ajax requests complete with an error. This is an Ajax Event.

Also in: [Ajax > Global Ajax Event Handlers](#)

---

## **.ajaxSend()**

Attach a function to be executed before an Ajax request is sent.  
This is an Ajax Event.

Also in: [Ajax > Global Ajax Event Handlers](#)

---

## **.ajaxStart()**

Register a handler to be called when the first Ajax request begins. This is an Ajax Event.

Also in: [Ajax > Global Ajax Event Handlers](#)

---

## **.ajaxStop()**

Register a handler to be called when all Ajax requests have completed. This is an Ajax Event.

Also in: [Ajax > Global Ajax Event Handlers](#)

---

## **.ajaxSuccess()**

Attach a function to be executed whenever an Ajax request completes successfully. This is an Ajax Event.

Also in: [Selectors](#) > [Basic](#)

---

## All Selector (“\*”)

Selects all elements.

Also in: [Traversing](#) > [Miscellaneous Traversing](#)

---

## **.andSelf()**

Add the previous set of elements on the stack to the current set.

Also in: [Effects](#) > [Custom](#)

---

## **.animate()**

Perform a custom animation of a set of CSS properties.

Also in: [Selectors > Basic Filter](#) | [Selectors > jQuery Extensions](#)

---

## **:animated Selector**

Select all elements that are in the progress of an animation at the time the selector is run.

Also in: [Manipulation](#) > [DOM Insertion, Inside](#)

---

## **.append()**

Insert content, specified by the parameter, to the end of each element in the set of matched elements.

Also in: [Manipulation](#) > [DOM Insertion, Inside](#)

---

## `.appendTo()`

Insert every element in the set of matched elements to the end of the target.

Also in: [Attributes](#) | [Manipulation](#) > [General Attributes](#)

---

## **.attr()**

Get the value of an attribute for the first element in the set of matched elements or set one or more attributes for every matched element.

Also in: [Selectors](#) > [Attribute](#)

---

## Attribute Contains Prefix Selector

### [name|="value"]

Selects elements that have the specified attribute with a value either equal to a given string or starting with that string followed by a hyphen (-).

Also in: [Selectors](#) > [Attribute](#)

---

## Attribute Contains Selector [name\*="value"]

Selects elements that have the specified attribute with a value containing the a given substring.

Also in: [Selectors](#) > [Attribute](#)

---

## Attribute Contains Word Selector

### `[name~="value"]`

Selects elements that have the specified attribute with a value containing a given word, delimited by spaces.

Also in: [Selectors](#) > [Attribute](#)

---

## Attribute Ends With Selector

### `[name$="value"]`

Selects elements that have the specified attribute with a value ending exactly with a given string. The comparison is case sensitive.

Also in: [Selectors](#) > [Attribute](#)

---

## Attribute Equals Selector [name="value"]

Selects elements that have the specified attribute with a value exactly equal to a certain value.

Also in: [Selectors > Attribute](#) | [Selectors > jQuery Extensions](#)

---

## Attribute Not Equal Selector

### [name!="value"]

Select elements that either don't have the specified attribute, or do have the specified attribute but not with a certain value.

Also in: [Selectors](#) > [Attribute](#)

---

## Attribute Starts With Selector

### `[name^="value"]`

Selects elements that have the specified attribute with a value beginning exactly with a given string.

Also in: [Manipulation](#) > [DOM Insertion, Outside](#)

---

## **.before()**

Insert content, specified by the parameter, before each element in the set of matched elements.

Also in: [Events](#) > [Event Handler Attachment](#)

---

## **.bind()**

Attach a handler to an event for the elements.

Also in: [Events > Form Events](#) | [Forms](#)

---

## **.blur()**

Bind an event handler to the “blur” JavaScript event, or trigger that event on an element.

Also in: [Selectors > Form](#) | [Selectors > jQuery Extensions](#)

---

## **:button Selector**

Selects all button elements and elements of type button.

Also in: [Callbacks Object](#)

---

## callbacks.add()

Add a callback or a collection of callbacks to a callback list.

Also in: [Callbacks Object](#)

---

## callbacks.disable()

Disable a callback list from doing anything more.

Also in: [Callbacks Object](#)

---

## callbacks.disabled()

Determine if the callbacks list has been disabled.

Also in: [Callbacks Object](#)

---

## callbacks.empty()

Remove all of the callbacks from a list.

Also in: [Callbacks Object](#)

---

## callbacks.fire()

Call all of the callbacks with the given arguments

Also in: [Callbacks Object](#)

---

## callbacks.fired()

Determine if the callbacks have already been called at least once.

Also in: [Callbacks Object](#)

---

## callbacks.fireWith()

Call all callbacks in a list with the given context and arguments.

Also in: [Callbacks Object](#)

---

## callbacks.has()

Determine whether a supplied callback is in a list

Also in: [Callbacks Object](#)

---

## callbacks.lock()

Lock a callback list in its current state.

Also in: [Callbacks Object](#)

---

## callbacks.locked()

Determine if the callbacks list has been locked.

Also in: [Callbacks Object](#)

---

## callbacks.remove()

Remove a callback or a collection of callbacks from a callback list.

Also in: [Events > Form Events](#) | [Forms](#)

---

## **.change()**

Bind an event handler to the “change” JavaScript event, or trigger that event on an element.

Also in: [Selectors > Form](#) | [Selectors > jQuery Extensions](#)

---

## **:checkbox Selector**

Selects all elements of type checkbox.

Also in: [Selectors](#) > [Form](#)

---

## **:checked Selector**

Matches all elements that are checked.

Also in: [Selectors](#) > [Hierarchy](#)

---

## `child")" href="0D4EEA27BD.htm" rel="bookmark">Child Selector (“parent > child”)`

Selects all direct child elements specified by “child” of elements specified by “parent”.

Also in: [Traversing](#) > [Tree Traversal](#)

---

## **.children()**

Get the children of each element in the set of matched elements, optionally filtered by a selector.

Also in: [Selectors](#) > [Basic](#)

---

## Class Selector (“`.class`”)

Selects all elements with the given class.

Also in: [Effects](#) > [Custom](#) | [Data](#) | [Utilities](#)

---

## **.clearQueue()**

Remove from the queue all items that have not yet been run.

Also in: [Events](#) > [Mouse Events](#)

---

## `.click()`

Bind an event handler to the “click” JavaScript event, or trigger that event on an element.

Also in: [Manipulation](#) > [Copying](#)

---

## `.clone()`

Create a deep copy of the set of matched elements.

Also in: [Traversing](#) > [Tree Traversal](#)

---

## `.closest()`

For each element in the set, get the first element that matches the selector by testing the element itself and traversing up through its ancestors in the DOM tree.

Also in: [Selectors](#) > [Content Filter](#)

---

## **:contains() Selector**

Select all elements that contain the specified text.

Also in: [Traversing](#) > [Miscellaneous Traversing](#)

---

## **.contents()**

Get the children of each element in the set of matched elements, including text and comment nodes.

Also in: [Internals](#) | [Properties](#) > [Properties of jQuery Object Instances](#)

---

## **.context**

The DOM node context originally passed to `jQuery()`; if none was passed then context will likely be the document.

Also in: [CSS](#) | [Manipulation > Style Properties](#)

---

## `.css()`

Get the value of a style property for the first element in the set of matched elements or set one or more CSS properties for every matched element.

Also in: [Data](#) | [Miscellaneous](#) > [Data Storage](#)

---

## **.data()**

Store arbitrary data associated with the matched elements or return the value at the named data store for the first element in the set of matched elements.

Also in: [Events](#) > [Mouse Events](#)

---

## **.dblclick()**

Bind an event handler to the “dblclick” JavaScript event, or trigger that event on an element.

Also in: [Deferred Object](#)

---

## deferred.always()

Add handlers to be called when the Deferred object is either resolved or rejected.

Also in: [Deferred Object](#)

---

## deferred.done()

Add handlers to be called when the Deferred object is resolved.

Also in: [Deferred Object](#)

---

## deferred.fail()

Add handlers to be called when the Deferred object is rejected.

Also in: [Deferred Object](#)

---

## deferred.isRejected()

Determine whether a Deferred object has been rejected.

Also in: [Deferred Object](#)

---

## deferred.isResolved()

Determine whether a Deferred object has been resolved.

Also in: [Deferred Object](#)

---

## deferred.notify()

Call the progressCallbacks on a Deferred object with the given args.

Also in: [Deferred Object](#)

---

## deferred.notifyWith()

Call the progressCallbacks on a Deferred object with the given context and args.

Also in: [Deferred Object](#)

---

## deferred.pipe()

Utility method to filter and/or chain Deferreds.

Also in: [Deferred Object](#)

---

## deferred.progress()

Add handlers to be called when the Deferred object generates progress notifications.

Also in: [Deferred Object](#)

---

## deferred.promise()

Return a Deferred's Promise object.

Also in: [Deferred Object](#)

---

## deferred.reject()

Reject a Deferred object and call any failCallbacks with the given args.

Also in: [Deferred Object](#)

---

## deferred.rejectWith()

Reject a Deferred object and call any failCallbacks with the given context and args.

Also in: [Deferred Object](#)

---

## deferred.resolve()

Resolve a Deferred object and call any doneCallbacks with the given args.

Also in: [Deferred Object](#)

---

## deferred.resolveWith()

Resolve a Deferred object and call any doneCallbacks with the given context and args.

Also in: [Deferred Object](#)

---

## **deferred.state()**

Determine the current state of a Deferred object.

Also in: [Deferred Object](#)

---

## deferred.then()

Add handlers to be called when the Deferred object is resolved, rejected, or still in progress.

Also in: [Effects](#) > [Custom](#)

---

## **.delay()**

Set a timer to delay execution of subsequent items in the queue.

Also in: [Events](#) > [Event Handler Attachment](#)

---

## **.delegate()**

Attach a handler to one or more events for all elements that match the selector, now or in the future, based on a specific set of root elements.

Also in: [Effects](#) > [Custom](#) | [Data](#) | [Utilities](#)

---

## **.dequeue()**

Execute the next function on the queue for the matched elements.

Also in: [Selectors](#) > [Hierarchy](#)

---

## Descendant Selector (“ancestor descendant”)

Selects all elements that are descendants of a given ancestor.

Also in: [Manipulation](#) > [DOM Removal](#)

---

---

## **.detach()**

Remove the set of matched elements from the DOM.

Also in: [Events](#) > [Event Handler Attachment](#)

---

## **.die()**

Remove event handlers previously attached using `.live()` from the elements.

Also in: [Selectors](#) > [Form](#)

---

## **:disabled Selector**

Selects all elements that are disabled.

Also in: [Miscellaneous](#) > [Collection Manipulation](#) | [Traversing](#)

---

## `.each()`

Iterate over a jQuery object, executing a function for each matched element.

Also in: [Selectors](#) > [Basic](#)

---

## Element Selector (“element”)

Selects all elements with the given tag name.

Also in: [Manipulation](#) > [DOM Removal](#)

---

## **.empty()**

Remove all child nodes of the set of matched elements from the DOM.

Also in: [Selectors > Content Filter](#)

---

## **:empty Selector**

Select all elements that have no children (including text nodes).

Also in: [Selectors](#) > [Form](#)

---

## **:enabled Selector**

Selects all elements that are enabled.

Also in: [Traversing](#) > [Miscellaneous Traversing](#)

---

## **.end()**

End the most recent filtering operation in the current chain and return the set of matched elements to its previous state.

Also in: [Traversing](#) > [Filtering](#)

---

## **.eq()**

Reduce the set of matched elements to the one at the specified index.

Also in: [Selectors > Basic Filter](#) | [Selectors > jQuery Extensions](#)

---

## **:eq() Selector**

Select the element at index n within the matched set.

Also in: [Events](#) > [Browser Events](#)

---

## **.error()**

Bind an event handler to the “error” JavaScript event.

Also in: [Selectors > Basic Filter](#) | [Selectors > jQuery Extensions](#)

---

## **:even Selector**

Selects even elements, zero-indexed. See also [odd](#).

Also in: [Events](#) > [Event Object](#)

---

## **event.currentTarget**

The current DOM element within the event bubbling phase.

Also in: [Events](#) > [Event Object](#)

---

## event.data

An optional object of data passed to an event method when the current executing handler is bound.

Also in: [Events](#) > [Event Object](#) | [Events](#)

---

## event.delegateTarget

The element where the currently-called jQuery event handler was attached.

Also in: [Events](#) > [Event Object](#)

---

## **event.isDefaultPrevented()**

Returns whether `event.preventDefault()` was ever called on this event object.

Also in: [Events](#) > [Event Object](#)

---

## **event.isImmediatePropagationStopped()**

Returns whether `event.stopImmediatePropagation()` was ever called on this event object.

Also in: [Events](#) > [Event Object](#)

---

## **event.isPropagationStopped()**

Returns whether `event.stopPropagation()` was ever called on this event object.

Also in: [Events](#) > [Event Object](#)

---

## event.metaKey

Indicates whether the META key was pressed when the event fired.

Also in: [Events](#) > [Event Object](#)

---

## event.namespace

The namespace specified when the event was triggered.

Also in: [Events](#) > [Event Object](#)

---

## event.pageX

The mouse position relative to the left edge of the document.

Also in: [Events](#) > [Event Object](#)

---

## event.pageY

The mouse position relative to the top edge of the document.

Also in: [Events](#) > [Event Object](#)

---

## **event.preventDefault()**

If this method is called, the default action of the event will not be triggered.

Also in: [Events > Event Object](#)

---

## **event.relatedTarget**

The other DOM element involved in the event, if any.

Also in: [Events](#) > [Event Object](#)

---

## **event.result**

The last value returned by an event handler that was triggered by this event, unless the value was undefined.

Also in: [Events](#) > [Event Object](#)

---

## **event.stopImmediatePropagation()**

Keeps the rest of the handlers from being executed and prevents the event from bubbling up the DOM tree.

Also in: [Events](#) > [Event Object](#)

---

## **event.stopPropagation()**

Prevents the event from bubbling up the DOM tree, preventing any parent handlers from being notified of the event.

Also in: [Events](#) > [Event Object](#)

---

## event.target

The DOM element that initiated the event.

Also in: [Events](#) > [Event Object](#)

---

## event.timeStamp

The difference in milliseconds between the time the browser created the event and January 1, 1970.

Also in: [Events](#) > [Event Object](#)

---

## event.type

Describes the nature of the event.

Also in: [Events](#) > [Event Object](#)

---

## event.which

For key or mouse events, this property indicates the specific key or button that was pressed.

Also in: [Effects > Fading](#)

---

## **.fadeIn()**

Display the matched elements by fading them to opaque.

Also in: [Effects](#) > [Fading](#)

---

## **.fadeOut()**

Hide the matched elements by fading them to transparent.

Also in: [Effects](#) > [Fading](#)

---

## `.fadeTo()`

Adjust the opacity of the matched elements.

Also in: [Effects](#) | [Effects > Fading](#)

---

## **.fadeToggle()**

Display or hide the matched elements by animating their opacity.

Also in: [Selectors > Form](#) | [Selectors > jQuery Extensions](#)

---

## **:file Selector**

Selects all elements of type file.

Also in: [Traversing](#) > [Filtering](#)

---

## **.filter()**

Reduce the set of matched elements to those that match the selector or pass the function's test.

Also in: [Traversing > Tree Traversal](#)

---

## **.find()**

Get the descendants of each element in the current set of matched elements, filtered by a selector, jQuery object, or element.

Also in: [Effects](#) > [Custom](#)

---

## **.finish()**

Stop the currently-running animation, remove all queued animations, and complete all animations for the matched elements.

Also in: [Traversing](#) > [Filtering](#)

---

## **.first()**

Reduce the set of matched elements to the first in the set.

Also in: [Selectors](#) > [Child Filter](#)

---

## **:first-child Selector**

Selects all elements that are the first child of their parent.

Also in: [Selectors](#) > [Child Filter](#)

---

## **:first-of-type Selector**

Selects all elements that are the first among siblings of the same element name.

Also in: [Selectors > Basic Filter](#) | [Selectors > jQuery Extensions](#)

---

## **:first Selector**

Selects the first matched element.

Also in: [Events > Form Events](#) | [Forms](#)

---

## **.focus()**

Bind an event handler to the “focus” JavaScript event, or trigger that event on an element.

Also in: [Selectors > Basic Filter](#) | [Selectors > Form](#)

---

## **:focus Selector**

Selects element if it is currently focused.

Also in: [Events > Keyboard Events](#) | [Events > Mouse Events](#)

---

## **.focusin()**

Bind an event handler to the “focusin” event.

Also in: [Events > Keyboard Events](#) | [Events > Mouse Events](#)

---

## **.focusout()**

Bind an event handler to the “focusout” JavaScript event.

Also in: [Miscellaneous](#) > [DOM Element Methods](#)

---

## `.get()`

Retrieve the DOM elements matched by the jQuery object.

Also in: [Selectors > Basic Filter](#) | [Selectors > jQuery Extensions](#)

---

## **:gt() Selector**

Select all elements at an index greater than index within the matched set.

Also in: [Traversing](#) > [Filtering](#)

---

## `.has()`

Reduce the set of matched elements to those that have a descendant that matches the selector or DOM element.

Also in: [Selectors](#) > [Attribute](#)

---

## Has Attribute Selector [name]

Selects elements that have the specified attribute, with any value.

Also in: [Selectors > Content Filter](#) | [Selectors > jQuery Extensions](#)

---

## **:has() Selector**

Selects elements which contain at least one element that matches the specified selector.

Also in: [Attributes](#) | [Manipulation > Class Attribute](#) | [CSS](#)

---

## **.hasClass()**

Determine whether any of the matched elements are assigned the given class.

Also in: [Selectors > Basic Filter](#) | [Selectors > jQuery Extensions](#)

---

## **:header Selector**

Selects all elements that are headers, like h1, h2, h3 and so on.

Also in: [CSS](#) | [Dimensions](#) | [Manipulation](#) > [Style Properties](#)

---

## **.height()**

Get the current computed height for the first element in the set of matched elements or set the height of every matched element.

Also in: [Selectors > jQuery Extensions](#) | [Selectors > Visibility Filter](#)

---

## **:hidden Selector**

Selects all elements that are hidden.

Also in: [Effects](#) > [Basics](#)

---

## **.hide()**

Hide the matched elements.

Also in: [Events](#) > [Mouse Events](#)

---

## **.hover()**

Bind one or two handlers to the matched elements, to be executed when the mouse pointer enters and leaves the elements.

Also in: [Attributes](#) | [Manipulation](#) > [DOM Insertion, Inside](#)

---

## `.html()`

Get the HTML contents of the first element in the set of matched elements or set the HTML contents of every matched element.

Also in: [Selectors](#) > [Basic](#)

---

## ID Selector (“#id”)

Selects a single element with the given id attribute.

Also in: [Selectors > Form](#) | [Selectors > jQuery Extensions](#)

---

## **:image Selector**

Selects all elements of type image.

Also in: [Miscellaneous](#) > [DOM Element Methods](#)

---

## `.index()`

Search for a given element from among the matched elements.

Also in: [CSS](#) | [Dimensions](#) | [Manipulation](#) > [Style Properties](#)

---

## **.innerHeight()**

Get the current computed height for the first element in the set of matched elements, including padding but not border.

Also in: [CSS](#) | [Dimensions](#) | [Manipulation > Style Properties](#)

---

## **.innerWidth()**

Get the current computed width for the first element in the set of matched elements, including padding but not border.

Also in: [Selectors > Form](#) | [Selectors > jQuery Extensions](#)

---

## **:input Selector**

Selects all input, textarea, select and button elements.

Also in: [Manipulation](#) > [DOM Insertion, Outside](#)

---

## **.insertAfter()**

Insert every element in the set of matched elements after the target.

Also in: [Manipulation > DOM Insertion, Outside](#)

---

## **.insertBefore()**

Insert every element in the set of matched elements before the target.

Also in: [Traversing](#) > [Filtering](#)

---

## `.is()`

Check the current matched set of elements against a selector, element, or jQuery object and return true if at least one of these elements matches the given arguments.

Also in: [Core](#)

---

## jQuery()

Return a collection of matched elements either found in the DOM based on passed argument(s) or created by passing an HTML string.

Also in: [Ajax](#) > [Low-Level Interface](#)

---

## jQuery.ajax()

Perform an asynchronous HTTP (Ajax) request.

Also in: [Ajax](#) > [Low-Level Interface](#)

---

## jQuery.ajaxPrefilter()

Handle custom Ajax options or modify existing options before each request is sent and before they are processed by \$.ajax().

Also in: [Ajax > Low-Level Interface](#)

---

## jQuery.ajaxSetup()

Set default values for future Ajax requests.

Also in: [Ajax](#) > [Low-Level Interface](#)

---

## jQuery.ajaxTransport()

Creates an object that handles the actual transmission of Ajax data.

Also in: [Utilities](#)

---

## jQuery.boxModel

Deprecated in jQuery 1.3 (see [jQuery.support](#)). States if the current page, in the user's browser, is being rendered using the W3C CSS Box Model.

Also in: [Properties](#) > [Properties of the Global jQuery Object](#) | [Utilities](#)

---

## jQuery.browser

Contains flags for the useragent, read from navigator.userAgent. We recommend against using this property; please try to use feature detection instead (see jQuery.support). jQuery.browser may be moved to a plugin in a future release of jQuery.

Also in: [Callbacks Object](#)

---

## jQuery.Callbacks()

A multi-purpose callbacks list object that provides a powerful way to manage callback lists.

Also in: [Utilities](#)

---

## jQuery.contains()

Check to see if a DOM element is a descendant of another DOM element.

Also in: [CSS](#)

---

## jQuery.cssHooks

Hook directly into jQuery to override how particular CSS properties are retrieved or set, normalize CSS property naming, or create custom properties.

Also in: [Data](#) | [Utilities](#)

---

## jQuery.data()

Store arbitrary data associated with the specified element and/or return the value that was set.

Also in: [Deferred Object](#)

---

## jQuery.Deferred()

A constructor function that returns a chainable utility object with methods to register multiple callbacks into callback queues, invoke callback queues, and relay the success or failure state of any synchronous or asynchronous function.

Also in: [Data](#) | [Utilities](#)

---

## jQuery.dequeue()

Execute the next function on the queue for the matched element.

Also in: [Utilities](#)

---

## jQuery.each()

A generic iterator function, which can be used to seamlessly iterate over both objects and arrays. Arrays and array-like objects with a length property (such as a function's arguments object) are iterated by numeric index, from 0 to length-1. Other objects are iterated via their named properties.

Also in: [Internals](#)

---

## jQuery.error()

Takes a string and throws an exception containing it.

Also in: [Utilities](#)

---

## jQuery.extend()

Merge the contents of two or more objects together into the first object.

Also in: [Effects > Custom](#) | [Properties > Properties of the Global jQuery Object](#)

---

## jQuery.fx.interval

The rate (in milliseconds) at which animations fire.

Also in: [Effects > Custom](#) | [Properties > Properties of the Global jQuery Object](#)

---

## jQuery.fx.off

Globally disable all animations.

Also in: [Ajax](#) > [Shorthand Methods](#)

---

## jQuery.get()

Load data from the server using a HTTP GET request.

Also in: [Ajax](#) > [Shorthand Methods](#)

---

## jQuery.getJSON()

Load JSON-encoded data from the server using a GET HTTP request.

Also in: [Ajax](#) > [Shorthand Methods](#)

---

## jQuery.getScript()

Load a JavaScript file from the server using a GET HTTP request, then execute it.

Also in: [Utilities](#)

---

## jQuery.globalEval()

Execute some JavaScript code globally.

Also in: [Utilities](#)

---

## jQuery.grep()

Finds the elements of an array which satisfy a filter function. The original array is not affected.

Also in: [Data](#)

---

## jQuery.hasData()

Determine whether an element has any jQuery data associated with it.

Also in: [Core](#)

---

## jQuery.holdReady()

Holds or releases the execution of jQuery's ready event.

Also in: [Utilities](#)

---

## jQuery.inArray()

Search for a specified value within an array and return its index (or -1 if not found).

Also in: [Utilities](#)

---

## jQuery.isArray()

Determine whether the argument is an array.

Also in: [Utilities](#)

---

## jQuery.isEmptyObject()

Check to see if an object is empty (contains no enumerable properties).

Also in: [Utilities](#)

---

## jQuery.isFunction()

Determine if the argument passed is a Javascript function object.

Also in: [Utilities](#)

---

## jQuery.isNumeric()

Determines whether its argument is a number.

Also in: [Utilities](#)

---

## jQuery.isPlainObject()

Check to see if an object is a plain object (created using “{}” or “new Object”).

Also in: [Utilities](#)

---

## jQuery.isWindow()

Determine whether the argument is a window.

Also in: [Utilities](#)

---

## jQuery.isXMLDoc()

Check to see if a DOM node is within an XML document (or is an XML document).

Also in: [Utilities](#)

---

## jQuery.makeArray()

Convert an array-like object into a true JavaScript array.

Also in: [Utilities](#)

---

## jQuery.map()

Translate all items in an array or object to new array of items.

Also in: [Utilities](#)

---

## jQuery.merge()

Merge the contents of two arrays together into the first array.

Also in: [Core](#) | [Miscellaneous](#) > [Setup Methods](#)

---

## jQuery.noConflict()

Relinquish jQuery's control of the \$ variable.

Also in: [Utilities](#)

---

## jQuery.noop()

An empty function.

Also in: [Utilities](#)

---

## jQuery.now()

Return a number representing the current time.

Also in: [Miscellaneous](#) > [Collection Manipulation](#) | [Forms](#) | [Ajax](#) > [Helper Functions](#)

---

## jQuery.param()

Create a serialized representation of an array or object, suitable for use in a URL query string or Ajax request.

Also in: [Utilities](#)

---

## jQuery.parseHTML()

Parses a string into an array of DOM nodes.

Also in: [Utilities](#)

---

## jQuery.parseJSON()

Takes a well-formed JSON string and returns the resulting JavaScript object.

Also in: [Utilities](#)

---

## jQuery.parseXML()

Parses a string into an XML document.

Also in: [Ajax](#) > [Shorthand Methods](#)

---

## jQuery.post()

Load data from the server using a HTTP POST request.

Also in: [Events](#) > [Event Handler Attachment](#) | [Utilities](#)

---

## jQuery.proxy()

Takes a function and returns a new one that will always have a particular context.

Also in: [Data](#) | [Utilities](#)

---

## jQuery.queue()

Show or manipulate the queue of functions to be executed on the matched element.

Also in: [Data](#) | [Utilities](#)

---

## jQuery.removeData()

Remove a previously-stored piece of data.

Also in: [Core](#)

---

## jQuery.sub()

Creates a new copy of jQuery whose properties and methods can be modified without affecting the original jQuery object.

Also in: [Properties](#) > [Properties of the Global jQuery Object](#) | [Utilities](#)

---

## jQuery.support

A collection of properties that represent the presence of different browser features or bugs. Primarily intended for jQuery's internal use; specific properties may be removed when they are no longer needed internally to improve page startup performance.

Also in: [Utilities](#)

---

## jQuery.trim()

Remove the whitespace from the beginning and end of a string.

Also in: [Utilities](#)

---

## jQuery.type()

Determine the internal JavaScript `[[Class]]` of an object.

Also in: [Utilities](#)

---

## jQuery.unique()

Sorts an array of DOM elements, in place, with the duplicates removed. Note that this only works on arrays of DOM elements, not strings or numbers.

Also in: [Core](#) | [Deferred Object](#)

---

## jQuery.when()

Provides a way to execute callback functions based on one or more objects, usually Deferred objects that represent asynchronous events.

Also in: [Events](#) > [Keyboard Events](#)

---

## **.keydown()**

Bind an event handler to the “keydown” JavaScript event, or trigger that event on an element.

Also in: [Events](#) > [Keyboard Events](#)

---

## **.keypress()**

Bind an event handler to the “keypress” JavaScript event, or trigger that event on an element.

Also in: [Events](#) > [Keyboard Events](#)

---

## **.keyup()**

Bind an event handler to the “keyup” JavaScript event, or trigger that event on an element.

Also in: [Selectors](#) > [Basic Filter](#)

---

## **:lang Selector**

Selects all elements of the specified language.

Also in: [Traversing](#) > [Filtering](#)

---

## **.last()**

Reduce the set of matched elements to the final one in the set.

Also in: [Selectors](#) > [Child Filter](#)

---

## **:last-child Selector**

Selects all elements that are the last child of their parent.

Also in: [Selectors](#) > [Child Filter](#)

---

## **:last-of-type Selector**

Selects all elements that are the last among siblings of the same element name.

Also in: [Selectors > Basic Filter](#) | [Selectors > jQuery Extensions](#)

---

## **:last Selector**

Selects the last matched element.

Also in: [Properties](#) > [Properties of jQuery Object Instances](#)

---

## **.length**

The number of elements in the jQuery object.

Also in: [Events](#) > [Event Handler Attachment](#)

---

## **.live()**

Attach an event handler for all elements which match the current selector, now and in the future.

Also in: [Ajax > Shorthand Methods](#)

---

## **.load()**

Load data from the server and place the returned HTML into the matched element.

Also in: [Events](#) > [Document Loading](#)

---

## **.load()**

Bind an event handler to the “load” JavaScript event.

Also in: [Selectors > Basic Filter](#) | [Selectors > jQuery Extensions](#)

---

## **:lt() Selector**

Select all elements at an index less than index within the matched set.

Also in: [Traversing](#) > [Filtering](#)

---

## **.map()**

Pass each element in the current matched set through a function, producing a new jQuery object containing the return values.

Also in: [Events](#) > [Mouse Events](#)

---

## **.mousedown()**

Bind an event handler to the “mousedown” JavaScript event, or trigger that event on an element.

Also in: [Events](#) > [Mouse Events](#)

---

## **.mouseenter()**

Bind an event handler to be fired when the mouse enters an element, or trigger that handler on an element.

Also in: [Events](#) > [Mouse Events](#)

---

## **.mouseleave()**

Bind an event handler to be fired when the mouse leaves an element, or trigger that handler on an element.

Also in: [Events](#) > [Mouse Events](#)

---

## `.mousemove()`

Bind an event handler to the “mousemove” JavaScript event, or trigger that event on an element.

Also in: [Events](#) > [Mouse Events](#)

---

## **.mouseout()**

Bind an event handler to the “mouseout” JavaScript event, or trigger that event on an element.

Also in: [Events](#) > [Mouse Events](#)

---

## `.mouseover()`

Bind an event handler to the “mouseover” JavaScript event, or trigger that event on an element.

Also in: [Events](#) > [Mouse Events](#)

---

## **.mouseup()**

Bind an event handler to the “mouseup” JavaScript event, or trigger that event on an element.

Also in: [Selectors](#) > [Attribute](#)

---

## Multiple Attribute Selector `[name="value"]` `[name2="value2"]`

Matches elements that match all of the specified attribute filters.

Also in: [Selectors](#) > [Basic](#)

---

---

## Multiple Selector (“selector1, selector2, selectorN”)

Selects the combined results of all the specified selectors.

Also in: [Traversing](#) > [Tree Traversal](#)

---

---

## **.next()**

Get the immediately following sibling of each element in the set of matched elements. If a selector is provided, it retrieves the next sibling only if it matches that selector.

Also in: [Selectors](#) > [Hierarchy](#)

---

## Next Adjacent Selector (“prev + next”)

Selects all next elements matching “next” that are immediately preceded by a sibling “prev”.

Also in: [Selectors > Hierarchy](#)

---

## Next Siblings Selector (“prev ~ siblings”)

Selects all sibling elements that follow after the “prev” element, have the same parent, and match the filtering “siblings” selector.

Also in: [Traversing > Tree Traversal](#)

---

## **.nextAll()**

Get all following siblings of each element in the set of matched elements, optionally filtered by a selector.

Also in: [Traversing > Tree Traversal](#)

---

## **.nextUntil()**

Get all following siblings of each element up to but not including the element matched by the selector, DOM node, or jQuery object passed.

Also in: [Traversing > Filtering](#) | [Traversing > Miscellaneous Traversing](#)

---

## **.not()**

Remove elements from the set of matched elements.

Also in: [Selectors](#) > [Basic Filter](#)

---

## **:not() Selector**

Selects all elements that do not match the given selector.

Also in: [Selectors](#) > [Child Filter](#)

---

## **:nth-child() Selector**

Selects all elements that are the nth-child of their parent.

Also in: [Selectors](#) > [Child Filter](#)

---

## **:nth-last-child() Selector**

Selects all elements that are the nth-child of their parent, counting from the last element to the first.

Also in: [Selectors](#) > [Child Filter](#)

---

## **:nth-last-of-type()** Selector

Selects all elements that are the nth-child of their parent, counting from the last element to the first.

Also in: [Selectors](#) > [Child Filter](#)

---

## **:nth-of-type() Selector**

Selects all elements that are the nth child of their parent in relation to siblings with the same element name.

Also in: [Selectors > Basic Filter](#) | [Selectors > jQuery Extensions](#)

---

## **:odd Selector**

Selects odd elements, zero-indexed. See also even.

Also in: [Events](#) > [Event Handler Attachment](#)

---

## **.off()**

Remove an event handler.

Also in: [CSS](#) | [Offset](#) | [Manipulation](#) > [Style Properties](#)

---

## **.offset()**

Get the current coordinates of the first element, or set the coordinates of every element, in the set of matched elements, relative to the document.

Also in: [Events](#) > [Event Handler Attachment](#)

---

## `.on()`

Attach an event handler function for one or more events to the selected elements.

Also in: [Events](#) > [Event Handler Attachment](#)

---

## `.one()`

Attach a handler to an event for the elements. The handler is executed at most once per element.

Also in: [Selectors](#) > [Child Filter](#)

---

## **:only-child Selector**

Selects all elements that are the only child of their parent.

Also in: [Selectors](#) > [Child Filter](#)

---

## **:only-of-type Selector**

Selects all elements that have no siblings with the same element name.

Also in: [CSS](#) | [Dimensions](#) | [Manipulation > Style Properties](#)

---

## **.outerHeight()**

Get the current computed height for the first element in the set of matched elements, including padding, border, and optionally margin. Returns an integer (without "px") representation of the value or null if called on an empty set of elements.

Also in: [CSS](#) | [Dimensions](#) | [Manipulation](#) > [Style Properties](#)

---

## **.outerWidth()**

Get the current computed width for the first element in the set of matched elements, including padding and border.

Also in: [Traversing > Tree Traversal](#)

---

## **.parent()**

Get the parent of each element in the current set of matched elements, optionally filtered by a selector.

Also in: [Selectors > Content Filter](#) | [Selectors > jQuery Extensions](#)

---

## **:parent Selector**

Select all elements that that have at least one child node (either an element or text).

Also in: [Traversing > Tree Traversal](#)

---

## **.parents()**

Get the ancestors of each element in the current set of matched elements, optionally filtered by a selector.

Also in: [Traversing](#) > [Tree Traversal](#)

---

## **.parentsUntil()**

Get the ancestors of each element in the current set of matched elements, up to but not including the element matched by the selector, DOM node, or jQuery object.

Also in: [Selectors > Form](#) | [Selectors > jQuery Extensions](#)

---

## **:password Selector**

Selects all elements of type password.

Also in: [CSS](#) | [Offset](#) | [Manipulation](#) > [Style Properties](#)

---

## **.position()**

Get the current coordinates of the first element in the set of matched elements, relative to the offset parent.

Also in: [Manipulation](#) > [DOM Insertion, Inside](#)

---

## **.prepend()**

Insert content, specified by the parameter, to the beginning of each element in the set of matched elements.

Also in: [Manipulation](#) > [DOM Insertion, Inside](#)

---

## **.prependTo()**

Insert every element in the set of matched elements to the beginning of the target.

Also in: [Traversing > Tree Traversal](#)

---

## **.prev()**

Get the immediately preceding sibling of each element in the set of matched elements, optionally filtered by a selector.

Also in: [Traversing > Tree Traversal](#)

---

## **.prevAll()**

Get all preceding siblings of each element in the set of matched elements, optionally filtered by a selector.

Also in: [Traversing](#) > [Tree Traversal](#)

---

## **.prevUntil()**

Get all preceding siblings of each element up to but not including the element matched by the selector, DOM node, or jQuery object.

Also in: [Deferred Object](#)

---

## **.promise()**

Return a Promise object to observe when all actions of a certain type bound to the collection, queued or not, have finished.

Also in: [Attributes](#) | [Manipulation](#) > [General Attributes](#)

---

## **.prop()**

Get the value of a property for the first element in the set of matched elements or set one or more properties for every matched element.

Also in: [Internals](#)

---

## **.pushStack()**

Add a collection of DOM elements onto the jQuery stack.

Also in: [Effects > Custom](#) | [Data](#) | [Utilities](#)

---

## **.queue()**

Show or manipulate the queue of functions to be executed on the matched elements.

Also in: [Selectors > Form](#) | [Selectors > jQuery Extensions](#)

---

## **:radio Selector**

Selects all elements of type radio.

Also in: [Events](#) > [Document Loading](#)

---

## **.ready()**

Specify a function to execute when the DOM is fully loaded.

Also in: [Manipulation](#) > [DOM Removal](#)

---

## `.remove()`

Remove the set of matched elements from the DOM.

Also in: [Attributes](#) | [Manipulation](#) > [General Attributes](#)

---

## `.removeAttr()`

Remove an attribute from each element in the set of matched elements.

Also in: [Attributes](#) | [Manipulation > Class Attribute](#) | [CSS](#)

---

## **.removeClass()**

Remove a single class, multiple classes, or all classes from each element in the set of matched elements.

Also in: [Data](#) | [Miscellaneous](#) > [Data Storage](#)

---

## **.removeData()**

Remove a previously-stored piece of data.

Also in: [Attributes](#) | [Manipulation](#) > [General Attributes](#)

---

## **.removeProp()**

Remove a property for the set of matched elements.

Also in: [Manipulation](#) > [DOM Replacement](#)

---

## `.replaceAll()`

Replace each target element with the set of matched elements.

Also in: [Manipulation](#) > [DOM Replacement](#)

---

## **.replaceWith()**

Replace each element in the set of matched elements with the provided new content and return the set of elements that was removed.

Also in: [Selectors > Form](#) | [Selectors > jQuery Extensions](#)

---

## **:reset Selector**

Selects all elements of type reset.

Also in: [Events](#) > [Browser Events](#)

---

## **.resize()**

Bind an event handler to the “resize” JavaScript event, or trigger that event on an element.

Also in: [Selectors](#) > [Basic Filter](#)

---

## **:root Selector**

Selects the element that is the root of the document.

Also in: [Events](#) > [Browser Events](#)

---

## `.scroll()`

Bind an event handler to the “scroll” JavaScript event, or trigger that event on an element.

Also in: [CSS](#) | [Offset](#) | [Manipulation](#) > [Style Properties](#)

---

## `.scrollLeft()`

Get the current horizontal position of the scroll bar for the first element in the set of matched elements or set the horizontal position of the scroll bar for every matched element.

Also in: [CSS](#) | [Offset](#) | [Manipulation](#) > [Style Properties](#)

---

## **.scrollTop()**

Get the current vertical position of the scroll bar for the first element in the set of matched elements or set the vertical position of the scroll bar for every matched element.

Also in: [Events > Form Events](#) | [Forms](#)

---

## **.select()**

Bind an event handler to the “select” JavaScript event, or trigger that event on an element.

Also in: [Selectors > Form](#) | [Selectors > jQuery Extensions](#)

---

## **:selected Selector**

Selects all elements that are selected.

Also in: [Forms](#) | [Ajax > Helper Functions](#)

---

## **.serialize()**

Encode a set of form elements as a string for submission.

Also in: [Forms](#) | [Ajax > Helper Functions](#)

---

## **.serializeArray()**

Encode a set of form elements as an array of names and values.

Also in: [Effects](#) > [Basics](#)

---

## `.show()`

Display the matched elements.

Also in: [Traversing](#) > [Tree Traversal](#)

---

## **.siblings()**

Get the siblings of each element in the set of matched elements, optionally filtered by a selector.

Also in: [Miscellaneous](#) > [DOM Element Methods](#)

---

## **.size()**

Return the number of elements in the jQuery object.

Also in: [Traversing](#) > [Filtering](#)

---

## **.slice()**

Reduce the set of matched elements to a subset specified by a range of indices.

Also in: [Effects](#) > [Sliding](#)

---

## **.slideDown()**

Display the matched elements with a sliding motion.

Also in: [Effects](#) > [Sliding](#)

---

## **.slideToggle()**

Display or hide the matched elements with a sliding motion.

Also in: [Effects](#) > [Sliding](#)

---

## `.slideUp()`

Hide the matched elements with a sliding motion.

Also in: [Effects](#) > [Custom](#)

---

## **.stop()**

Stop the currently-running animation on the matched elements.

Also in: [Events > Form Events](#) | [Forms](#)

---

## **.submit()**

Bind an event handler to the “submit” JavaScript event, or trigger that event on an element.

Also in: [Selectors > Form](#) | [Selectors > jQuery Extensions](#)

---

## **:submit Selector**

Selects all elements of type submit.

Also in: [Selectors](#) > [Basic Filter](#)

---

## **:target Selector**

Selects the target element indicated by the fragment identifier of the document's URI.

Also in: [Manipulation](#) > [DOM Insertion, Inside](#)

---

## **.text()**

Get the combined text contents of each element in the set of matched elements, including their descendants, or set the text contents of the matched elements.

Also in: [Selectors > Form](#) | [Selectors > jQuery Extensions](#)

---

## **:text Selector**

Selects all elements of type text.

Also in: [Miscellaneous](#) > [DOM Element Methods](#)

---

## **.toArray()**

Retrieve all the DOM elements contained in the jQuery set, as an array.

Also in: [Effects](#) > [Basics](#)

---

## **.toggle()**

Display or hide the matched elements.

Also in: [Deprecated](#) > [Deprecated 1.8](#) | [Events](#) > [Mouse Events](#)

---

## **.toggle()**

Bind two or more handlers to the matched elements, to be executed on alternate clicks.

Also in: [Attributes](#) | [Manipulation > Class Attribute](#) | [CSS](#)

---

## **.toggleClass()**

Add or remove one or more classes from each element in the set of matched elements, depending on either the class's presence or the value of the switch argument.

Also in: [Events](#) > [Event Handler Attachment](#)

---

## **.trigger()**

Execute all handlers and behaviors attached to the matched elements for the given event type.

Also in: [Events](#) > [Event Handler Attachment](#)

---

## **.triggerHandler()**

Execute all handlers attached to an element for an event.

Also in: [Events](#) > [Event Handler Attachment](#)

---

## **.unbind()**

Remove a previously-attached event handler from the elements.

Also in: [Events](#) > [Event Handler Attachment](#)

---

## **.undelegate()**

Remove a handler from the event for all elements which match the current selector, based upon a specific set of root elements.

Also in: [Events](#) > [Document Loading](#)

---

## **.unload()**

Bind an event handler to the “unload” JavaScript event.

Also in: [Manipulation > DOM Insertion, Around](#) | [Manipulation > DOM Removal](#)

---

## `.unwrap()`

Remove the parents of the set of matched elements from the DOM, leaving the matched elements in their place.

Also in: [Attributes](#) | [Forms](#) | [Manipulation](#) > [General Attributes](#)

---

## **.val()**

Get the current value of the first element in the set of matched elements or set the value of every matched element.

Also in: [Selectors > jQuery Extensions](#) | [Selectors > Visibility Filter](#)

---

## **:visible Selector**

Selects all elements that are visible.

Also in: [CSS](#) | [Dimensions](#) | [Manipulation](#) > [Style Properties](#)

---

## **.width()**

Get the current computed width for the first element in the set of matched elements or set the width of every matched element.

Also in: [Manipulation](#) > [DOM Insertion, Around](#)

---

## `.wrap()`

Wrap an HTML structure around each element in the set of matched elements.

Also in: [Manipulation](#) > [DOM Insertion, Around](#)

---

## **.wrapAll()**

Wrap an HTML structure around all elements in the set of matched elements.

Also in: [Manipulation](#) > [DOM Insertion, Around](#)

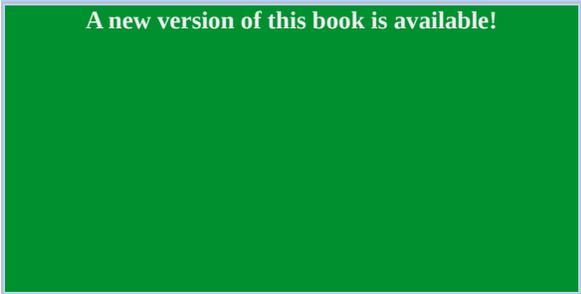
---

## **.wrapInner()**

Wrap an HTML structure around the content of each element in the set of matched elements.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

## Category: Version 1.0

All the aspects of the API that were added, or had a new signature added, in the corresponding version of jQuery.

[jQuery 1.0 Release Notes](#).

Also in: [Traversing](#) > [Miscellaneous Traversing](#)

---

## **.add()**

Add elements to the set of matched elements.

Also in: [Attributes](#) | [Manipulation > Class Attribute](#) | [CSS](#)

---

## **.addClass()**

Adds the specified class(es) to each of the set of matched elements.

Also in: [Manipulation > DOM Insertion, Outside](#)

---

## **.after()**

Insert content, specified by the parameter, after each element in the set of matched elements.

Also in: [Ajax > Global Ajax Event Handlers](#)

---

## **.ajaxComplete()**

Register a handler to be called when Ajax requests complete. This is an AjaxEvent.

Also in: [Ajax > Global Ajax Event Handlers](#)

---

## **.ajaxError()**

Register a handler to be called when Ajax requests complete with an error. This is an Ajax Event.

Also in: [Ajax > Global Ajax Event Handlers](#)

---

## **.ajaxSend()**

Attach a function to be executed before an Ajax request is sent.  
This is an Ajax Event.

Also in: [Ajax > Global Ajax Event Handlers](#)

---

## **.ajaxStart()**

Register a handler to be called when the first Ajax request begins. This is an Ajax Event.

Also in: [Ajax > Global Ajax Event Handlers](#)

---

## **.ajaxStop()**

Register a handler to be called when all Ajax requests have completed. This is an Ajax Event.

Also in: [Ajax > Global Ajax Event Handlers](#)

---

## **.ajaxSuccess()**

Attach a function to be executed whenever an Ajax request completes successfully. This is an Ajax Event.

Also in: [Selectors](#) > [Basic](#)

---

## All Selector (“\*”)

Selects all elements.

Also in: [Effects](#) > [Custom](#)

---

## **.animate()**

Perform a custom animation of a set of CSS properties.

Also in: [Manipulation](#) > [DOM Insertion, Inside](#)

---

## **.append()**

Insert content, specified by the parameter, to the end of each element in the set of matched elements.

Also in: [Manipulation](#) > [DOM Insertion, Inside](#)

---

## **.appendTo()**

Insert every element in the set of matched elements to the end of the target.

Also in: [Attributes](#) | [Manipulation](#) > [General Attributes](#)

---

## **.attr()**

Get the value of an attribute for the first element in the set of matched elements or set one or more attributes for every matched element.

Also in: [Selectors](#) > [Attribute](#)

---

## Attribute Contains Prefix Selector

### `[name="value"]`

Selects elements that have the specified attribute with a value either equal to a given string or starting with that string followed by a hyphen (-).

Also in: [Selectors](#) > [Attribute](#)

---

## Attribute Contains Selector [name\*="value"]

Selects elements that have the specified attribute with a value containing the a given substring.

Also in: [Selectors](#) > [Attribute](#)

---

## Attribute Contains Word Selector

### `[name~="value"]`

Selects elements that have the specified attribute with a value containing a given word, delimited by spaces.

Also in: [Selectors](#) > [Attribute](#)

---

## Attribute Ends With Selector

### `[name$="value"]`

Selects elements that have the specified attribute with a value ending exactly with a given string. The comparison is case sensitive.

Also in: [Selectors](#) > [Attribute](#)

---

## Attribute Equals Selector [name="value"]

Selects elements that have the specified attribute with a value exactly equal to a certain value.

Also in: [Selectors > Attribute](#) | [Selectors > jQuery Extensions](#)

---

## Attribute Not Equal Selector

### [name!="value"]

Select elements that either don't have the specified attribute, or do have the specified attribute but not with a certain value.

Also in: [Selectors](#) > [Attribute](#)

---

## Attribute Starts With Selector

### `[name^="value"]`

Selects elements that have the specified attribute with a value beginning exactly with a given string.

Also in: [Manipulation](#) > [DOM Insertion, Outside](#)

---

## **.before()**

Insert content, specified by the parameter, before each element in the set of matched elements.

Also in: [Events](#) > [Event Handler Attachment](#)

---

## **.bind()**

Attach a handler to an event for the elements.

Also in: [Events > Form Events](#) | [Forms](#)

---

## **.blur()**

Bind an event handler to the “blur” JavaScript event, or trigger that event on an element.

Also in: [Selectors > Form](#) | [Selectors > jQuery Extensions](#)

---

## **:button Selector**

Selects all button elements and elements of type button.

Also in: [Events > Form Events](#) | [Forms](#)

---

## **.change()**

Bind an event handler to the “change” JavaScript event, or trigger that event on an element.

Also in: [Selectors > Form](#) | [Selectors > jQuery Extensions](#)

---

## **:checkbox Selector**

Selects all elements of type checkbox.

Also in: [Selectors](#) > [Form](#)

---

## **:checked Selector**

Matches all elements that are checked.

Also in: [Selectors](#) > [Hierarchy](#)

---

## `child")" href="0D4EEA27BD.htm" rel="bookmark">Child Selector (“parent > child”)`

Selects all direct child elements specified by “child” of elements specified by “parent”.

Also in: [Traversing](#) > [Tree Traversal](#)

---

## **.children()**

Get the children of each element in the set of matched elements, optionally filtered by a selector.

Also in: [Selectors](#) > [Basic](#)

---

## Class Selector (“`.class`”)

Selects all elements with the given class.

Also in: [Events](#) > [Mouse Events](#)

---

## `.click()`

Bind an event handler to the “click” JavaScript event, or trigger that event on an element.

Also in: [Manipulation](#) > [Copying](#)

---

## `.clone()`

Create a deep copy of the set of matched elements.

Also in: [CSS](#) | [Manipulation](#) > [Style Properties](#)

---

## `.css()`

Get the value of a style property for the first element in the set of matched elements or set one or more CSS properties for every matched element.

Also in: [Events](#) > [Mouse Events](#)

---

## **.dblclick()**

Bind an event handler to the “dblclick” JavaScript event, or trigger that event on an element.

Also in: [Selectors > Hierarchy](#)

---

## Descendant Selector (“ancestor descendant”)

Selects all elements that are descendants of a given ancestor.

Also in: [Selectors](#) > [Form](#)

---

---

## **:disabled Selector**

Selects all elements that are disabled.

Also in: [Miscellaneous](#) > [Collection Manipulation](#) | [Traversing](#)

---

## `.each()`

Iterate over a jQuery object, executing a function for each matched element.

Also in: [Selectors](#) > [Basic](#)

---

## Element Selector (“element”)

Selects all elements with the given tag name.

Also in: [Manipulation](#) > [DOM Removal](#)

---

## **.empty()**

Remove all child nodes of the set of matched elements from the DOM.

Also in: [Selectors > Content Filter](#)

---

## **:empty Selector**

Select all elements that have no children (including text nodes).

Also in: [Selectors](#) > [Form](#)

---

## **:enabled Selector**

Selects all elements that are enabled.

Also in: [Traversing](#) > [Miscellaneous Traversing](#)

---

## **.end()**

End the most recent filtering operation in the current chain and return the set of matched elements to its previous state.

Also in: [Selectors > Basic Filter](#) | [Selectors > jQuery Extensions](#)

---

## **:eq() Selector**

Select the element at index n within the matched set.

Also in: [Events](#) > [Browser Events](#)

---

## **.error()**

Bind an event handler to the “error” JavaScript event.

Also in: [Selectors > Basic Filter](#) | [Selectors > jQuery Extensions](#)

---

## **:even Selector**

Selects even elements, zero-indexed. See also [odd](#).

Also in: [Events](#) > [Event Object](#)

---

## **event.preventDefault()**

If this method is called, the default action of the event will not be triggered.

Also in: [Events](#) > [Event Object](#)

---

## **event.stopPropagation()**

Prevents the event from bubbling up the DOM tree, preventing any parent handlers from being notified of the event.

Also in: [Events](#) > [Event Object](#)

---

## event.target

The DOM element that initiated the event.

Also in: [Events](#) > [Event Object](#)

---

## event.type

Describes the nature of the event.

Also in: [Effects](#) > [Fading](#)

---

## **.fadeIn()**

Display the matched elements by fading them to opaque.

Also in: [Effects](#) > [Fading](#)

---

## **.fadeOut()**

Hide the matched elements by fading them to transparent.

Also in: [Effects](#) > [Fading](#)

---

## **.fadeTo()**

Adjust the opacity of the matched elements.

Also in: [Selectors > Form](#) | [Selectors > jQuery Extensions](#)

---

## **:file Selector**

Selects all elements of type file.

Also in: [Traversing](#) > [Filtering](#)

---

## **.filter()**

Reduce the set of matched elements to those that match the selector or pass the function's test.

Also in: [Traversing > Tree Traversal](#)

---

## **.find()**

Get the descendants of each element in the current set of matched elements, filtered by a selector, jQuery object, or element.

Also in: [Selectors > Basic Filter](#) | [Selectors > jQuery Extensions](#)

---

## **:first Selector**

Selects the first matched element.

Also in: [Events > Form Events](#) | [Forms](#)

---

## **.focus()**

Bind an event handler to the “focus” JavaScript event, or trigger that event on an element.

Also in: [Miscellaneous](#) > [DOM Element Methods](#)

---

## `.get()`

Retrieve the DOM elements matched by the jQuery object.

Also in: [Selectors > Basic Filter](#) | [Selectors > jQuery Extensions](#)

---

## **:gt() Selector**

Select all elements at an index greater than index within the matched set.

Also in: [Selectors](#) > [Attribute](#)

---

## Has Attribute Selector [name]

Selects elements that have the specified attribute, with any value.

Also in: [CSS](#) | [Dimensions](#) | [Manipulation](#) > [Style Properties](#)

---

## **.height()**

Get the current computed height for the first element in the set of matched elements or set the height of every matched element.

Also in: [Selectors > jQuery Extensions](#) | [Selectors > Visibility Filter](#)

---

## **:hidden Selector**

Selects all elements that are hidden.

Also in: [Effects](#) > [Basics](#)

---

## **.hide()**

Hide the matched elements.

Also in: [Events](#) > [Mouse Events](#)

---

## **.hover()**

Bind one or two handlers to the matched elements, to be executed when the mouse pointer enters and leaves the elements.

Also in: [Attributes](#) | [Manipulation](#) > [DOM Insertion, Inside](#)

---

## `.html()`

Get the HTML contents of the first element in the set of matched elements or set the HTML contents of every matched element.

Also in: [Selectors](#) > [Basic](#)

---

## ID Selector (“#id”)

Selects a single element with the given id attribute.

Also in: [Selectors > Form](#) | [Selectors > jQuery Extensions](#)

---

## **:image Selector**

Selects all elements of type image.

Also in: [Miscellaneous](#) > [DOM Element Methods](#)

---

## `.index()`

Search for a given element from among the matched elements.

Also in: [Selectors > Form](#) | [Selectors > jQuery Extensions](#)

---

## **:input Selector**

Selects all input, textarea, select and button elements.

Also in: [Manipulation](#) > [DOM Insertion, Outside](#)

---

## **.insertAfter()**

Insert every element in the set of matched elements after the target.

Also in: [Manipulation > DOM Insertion, Outside](#)

---

## **.insertBefore()**

Insert every element in the set of matched elements before the target.

Also in: [Traversing](#) > [Filtering](#)

---

## `.is()`

Check the current matched set of elements against a selector, element, or jQuery object and return true if at least one of these elements matches the given arguments.

Also in: [Core](#)

---

## jQuery()

Return a collection of matched elements either found in the DOM based on passed argument(s) or created by passing an HTML string.

Also in: [Ajax](#) > [Low-Level Interface](#)

---

## jQuery.ajax()

Perform an asynchronous HTTP (Ajax) request.

Also in: [Utilities](#)

---

## jQuery.boxModel

Deprecated in jQuery 1.3 (see [jQuery.support](#)). States if the current page, in the user's browser, is being rendered using the W3C CSS Box Model.

Also in: [Properties](#) > [Properties of the Global jQuery Object](#) | [Utilities](#)

---

## jQuery.browser

Contains flags for the useragent, read from navigator.userAgent. We recommend against using this property; please try to use feature detection instead (see jQuery.support). jQuery.browser may be moved to a plugin in a future release of jQuery.

Also in: [Utilities](#)

---

## jQuery.each()

A generic iterator function, which can be used to seamlessly iterate over both objects and arrays. Arrays and array-like objects with a length property (such as a function's arguments object) are iterated by numeric index, from 0 to length-1. Other objects are iterated via their named properties.

Also in: [Utilities](#)

---

## jQuery.extend()

Merge the contents of two or more objects together into the first object.

Also in: [Ajax](#) > [Shorthand Methods](#)

---

## jQuery.get()

Load data from the server using a HTTP GET request.

Also in: [Ajax](#) > [Shorthand Methods](#)

---

## jQuery.getJSON()

Load JSON-encoded data from the server using a GET HTTP request.

Also in: [Ajax > Shorthand Methods](#)

---

## jQuery.getScript()

Load a JavaScript file from the server using a GET HTTP request, then execute it.

Also in: [Utilities](#)

---

## jQuery.grep()

Finds the elements of an array which satisfy a filter function. The original array is not affected.

Also in: [Utilities](#)

---

## jQuery.map()

Translate all items in an array or object to new array of items.

Also in: [Utilities](#)

---

## jQuery.merge()

Merge the contents of two arrays together into the first array.

Also in: [Core](#) | [Miscellaneous](#) > [Setup Methods](#)

---

## jQuery.noConflict()

Relinquish jQuery's control of the \$ variable.

Also in: [Ajax](#) > [Shorthand Methods](#)

---

## jQuery.post()

Load data from the server using a HTTP POST request.

Also in: [Utilities](#)

---

## jQuery.trim()

Remove the whitespace from the beginning and end of a string.

Also in: [Events](#) > [Keyboard Events](#)

---

## **.keydown()**

Bind an event handler to the “keydown” JavaScript event, or trigger that event on an element.

Also in: [Events](#) > [Keyboard Events](#)

---

## **.keypress()**

Bind an event handler to the “keypress” JavaScript event, or trigger that event on an element.

Also in: [Events](#) > [Keyboard Events](#)

---

## **.keyup()**

Bind an event handler to the “keyup” JavaScript event, or trigger that event on an element.

Also in: [Selectors > Basic Filter](#) | [Selectors > jQuery Extensions](#)

---

## **:last Selector**

Selects the last matched element.

Also in: [Properties](#) > [Properties of jQuery Object Instances](#)

---

## **.length**

The number of elements in the jQuery object.

Also in: [Ajax](#) > [Shorthand Methods](#)

---

## **.load()**

Load data from the server and place the returned HTML into the matched element.

Also in: [Events](#) > [Document Loading](#)

---

## **.load()**

Bind an event handler to the “load” JavaScript event.

Also in: [Selectors > Basic Filter](#) | [Selectors > jQuery Extensions](#)

---

## **:lt()** Selector

Select all elements at an index less than index within the matched set.

Also in: [Events](#) > [Mouse Events](#)

---

## **.mousedown()**

Bind an event handler to the “mousedown” JavaScript event, or trigger that event on an element.

Also in: [Events](#) > [Mouse Events](#)

---

## **.mouseenter()**

Bind an event handler to be fired when the mouse enters an element, or trigger that handler on an element.

Also in: [Events](#) > [Mouse Events](#)

---

## **.mouseleave()**

Bind an event handler to be fired when the mouse leaves an element, or trigger that handler on an element.

Also in: [Events](#) > [Mouse Events](#)

---

## `.mousemove()`

Bind an event handler to the “mousemove” JavaScript event, or trigger that event on an element.

Also in: [Events](#) > [Mouse Events](#)

---

## **.mouseout()**

Bind an event handler to the “mouseout” JavaScript event, or trigger that event on an element.

Also in: [Events](#) > [Mouse Events](#)

---

## `.mouseover()`

Bind an event handler to the “mouseover” JavaScript event, or trigger that event on an element.

Also in: [Events](#) > [Mouse Events](#)

---

## **.mouseup()**

Bind an event handler to the “mouseup” JavaScript event, or trigger that event on an element.

Also in: [Selectors](#) > [Attribute](#)

---

## Multiple Attribute Selector `[name="value"]` `[name2="value2"]`

Matches elements that match all of the specified attribute filters.

Also in: [Selectors](#) > [Basic](#)

---

---

## Multiple Selector (“selector1, selector2, selectorN”)

Selects the combined results of all the specified selectors.

Also in: [Traversing](#) > [Tree Traversal](#)

---

---

## **.next()**

Get the immediately following sibling of each element in the set of matched elements. If a selector is provided, it retrieves the next sibling only if it matches that selector.

Also in: [Selectors](#) > [Hierarchy](#)

---

## Next Adjacent Selector (“prev + next”)

Selects all next elements matching “next” that are immediately preceded by a sibling “prev”.

Also in: [Selectors > Hierarchy](#)

---

## Next Siblings Selector (“prev ~ siblings”)

Selects all sibling elements that follow after the “prev” element, have the same parent, and match the filtering “siblings” selector.

Also in: [Traversing > Filtering](#) | [Traversing > Miscellaneous Traversing](#)

---

## **.not()**

Remove elements from the set of matched elements.

Also in: [Selectors](#) > [Basic Filter](#)

---

## **:not() Selector**

Selects all elements that do not match the given selector.

Also in: [Selectors > Basic Filter](#) | [Selectors > jQuery Extensions](#)

---

## **:odd Selector**

Selects odd elements, zero-indexed. See also even.

Also in: [Traversing](#) > [Tree Traversal](#)

---

## **.parent()**

Get the parent of each element in the current set of matched elements, optionally filtered by a selector.

Also in: [Selectors > Content Filter](#) | [Selectors > jQuery Extensions](#)

---

## **:parent Selector**

Select all elements that that have at least one child node (either an element or text).

Also in: [Traversing > Tree Traversal](#)

---

## **.parents()**

Get the ancestors of each element in the current set of matched elements, optionally filtered by a selector.

Also in: [Selectors > Form](#) | [Selectors > jQuery Extensions](#)

---

## **:password Selector**

Selects all elements of type password.

Also in: [Manipulation](#) > [DOM Insertion, Inside](#)

---

## **.prepend()**

Insert content, specified by the parameter, to the beginning of each element in the set of matched elements.

Also in: [Manipulation](#) > [DOM Insertion, Inside](#)

---

## **.prependTo()**

Insert every element in the set of matched elements to the beginning of the target.

Also in: [Traversing > Tree Traversal](#)

---

## **.prev()**

Get the immediately preceding sibling of each element in the set of matched elements, optionally filtered by a selector.

Also in: [Internals](#)

---

## **.pushStack()**

Add a collection of DOM elements onto the jQuery stack.

Also in: [Selectors > Form](#) | [Selectors > jQuery Extensions](#)

---

## **:radio Selector**

Selects all elements of type radio.

Also in: [Events](#) > [Document Loading](#)

---

## **.ready()**

Specify a function to execute when the DOM is fully loaded.

Also in: [Manipulation](#) > [DOM Removal](#)

---

## `.remove()`

Remove the set of matched elements from the DOM.

Also in: [Attributes](#) | [Manipulation](#) > [General Attributes](#)

---

## `.removeAttr()`

Remove an attribute from each element in the set of matched elements.

Also in: [Attributes](#) | [Manipulation > Class Attribute](#) | [CSS](#)

---

## **.removeClass()**

Remove a single class, multiple classes, or all classes from each element in the set of matched elements.

Also in: [Selectors > Form](#) | [Selectors > jQuery Extensions](#)

---

## **:reset Selector**

Selects all elements of type reset.

Also in: [Events](#) > [Browser Events](#)

---

## **.resize()**

Bind an event handler to the “resize” JavaScript event, or trigger that event on an element.

Also in: [Events](#) > [Browser Events](#)

---

## **.scroll()**

Bind an event handler to the “scroll” JavaScript event, or trigger that event on an element.

Also in: [Events > Form Events](#) | [Forms](#)

---

## **.select()**

Bind an event handler to the “select” JavaScript event, or trigger that event on an element.

Also in: [Selectors > Form](#) | [Selectors > jQuery Extensions](#)

---

## **:selected Selector**

Selects all elements that are selected.

Also in: [Forms](#) | [Ajax > Helper Functions](#)

---

## **.serialize()**

Encode a set of form elements as a string for submission.

Also in: [Effects](#) > [Basics](#)

---

## `.show()`

Display the matched elements.

Also in: [Traversing](#) > [Tree Traversal](#)

---

## **.siblings()**

Get the siblings of each element in the set of matched elements, optionally filtered by a selector.

Also in: [Miscellaneous](#) > [DOM Element Methods](#)

---

## **.size()**

Return the number of elements in the jQuery object.

Also in: [Effects](#) > [Sliding](#)

---

## **.slideDown()**

Display the matched elements with a sliding motion.

Also in: [Effects](#) > [Sliding](#)

---

## **.slideToggle()**

Display or hide the matched elements with a sliding motion.

Also in: [Effects](#) > [Sliding](#)

---

## `.slideUp()`

Hide the matched elements with a sliding motion.

Also in: [Events > Form Events](#) | [Forms](#)

---

## **.submit()**

Bind an event handler to the “submit” JavaScript event, or trigger that event on an element.

Also in: [Selectors > Form](#) | [Selectors > jQuery Extensions](#)

---

## **:submit Selector**

Selects all elements of type submit.

Also in: [Manipulation](#) > [DOM Insertion, Inside](#)

---

## **.text()**

Get the combined text contents of each element in the set of matched elements, including their descendants, or set the text contents of the matched elements.

Also in: [Selectors > Form](#) | [Selectors > jQuery Extensions](#)

---

## **:text Selector**

Selects all elements of type text.

Also in: [Effects](#) > [Basics](#)

---

## **.toggle()**

Display or hide the matched elements.

Also in: [Deprecated](#) > [Deprecated 1.8](#) | [Events](#) > [Mouse Events](#)

---

## **.toggle()**

Bind two or more handlers to the matched elements, to be executed on alternate clicks.

Also in: [Attributes](#) | [Manipulation > Class Attribute](#) | [CSS](#)

---

## **.toggleClass()**

Add or remove one or more classes from each element in the set of matched elements, depending on either the class's presence or the value of the switch argument.

Also in: [Events](#) > [Event Handler Attachment](#)

---

## **.trigger()**

Execute all handlers and behaviors attached to the matched elements for the given event type.

Also in: [Events](#) > [Event Handler Attachment](#)

---

## **.unbind()**

Remove a previously-attached event handler from the elements.

Also in: [Events](#) > [Document Loading](#)

---

## **.unload()**

Bind an event handler to the “unload” JavaScript event.

Also in: [Attributes](#) | [Forms](#) | [Manipulation](#) > [General Attributes](#)

---

## **.val()**

Get the current value of the first element in the set of matched elements or set the value of every matched element.

Also in: [Selectors > jQuery Extensions](#) | [Selectors > Visibility Filter](#)

---

## **:visible Selector**

Selects all elements that are visible.

Also in: [CSS](#) | [Dimensions](#) | [Manipulation](#) > [Style Properties](#)

---

## **.width()**

Get the current computed width for the first element in the set of matched elements or set the width of every matched element.

Also in: [Manipulation](#) > [DOM Insertion, Around](#)

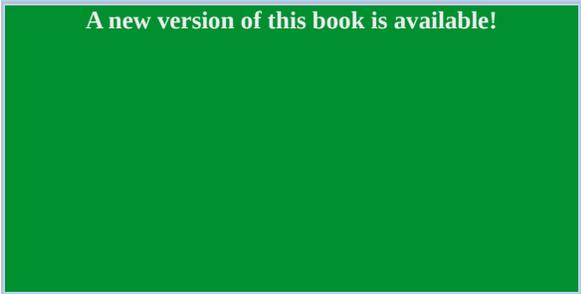
---

## **.wrap()**

Wrap an HTML structure around each element in the set of matched elements.

POWERED BY **HERONOTE**

A new version of this book is available!



---

## Category: Version 1.0.4

All the aspects of the API that were added, or had a new signature added, in the corresponding version of jQuery.

Release Notes: [1.0.1](#), [1.0.2](#), [1.0.3](#), [1.0.4](#).

Also in: [Events](#) > [Event Object](#)

---

## event.metaKey

Indicates whether the META key was pressed when the event fired.

Also in: [Events](#) > [Event Object](#)

---

## event.pageX

The mouse position relative to the left edge of the document.

Also in: [Events](#) > [Event Object](#)

---

## event.pageY

The mouse position relative to the top edge of the document.

Also in: [Utilities](#)

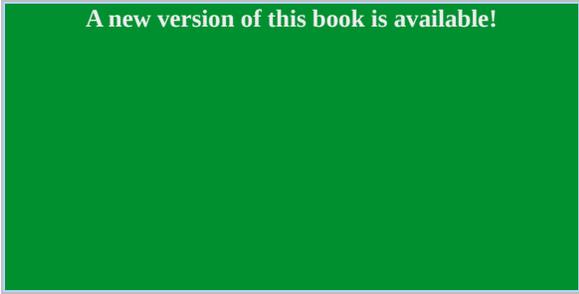
---

## jQuery.globalEval()

Execute some JavaScript code globally.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

# Category: Version 1.1

All the aspects of the API that were added, or had a new signature added, in the corresponding version of jQuery.

[jQuery 1.1 Release Notes](#).

Also in: [Attributes](#) | [Manipulation](#) > [General Attributes](#)

---

## **.attr()**

Get the value of an attribute for the first element in the set of matched elements or set one or more attributes for every matched element.

Also in: [Events](#) > [Event Object](#)

---

## event.data

An optional object of data passed to an event method when the current executing handler is bound.

Also in: [Ajax > Low-Level Interface](#)

---

## jQuery.ajaxSetup()

Set default values for future Ajax requests.

Also in: [Events](#) > [Event Handler Attachment](#)

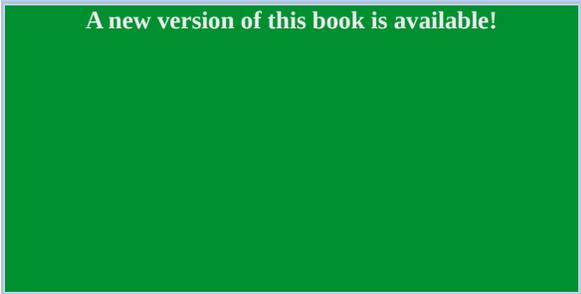
---

## `.one()`

Attach a handler to an event for the elements. The handler is executed at most once per element.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

## Category: Version 1.1.2

All the aspects of the API that were added, or had a new signature added, in the corresponding version of jQuery.

[jQuery 1.1.2 Release Notes](#).

Also in: [Traversing](#) > [Filtering](#)

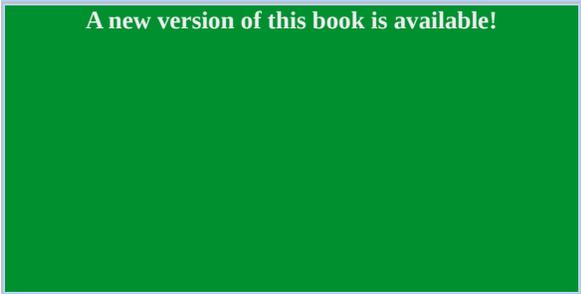
---

**.eq()**

Reduce the set of matched elements to the one at the specified index.

POWERED BY **HERONOTE**

A new version of this book is available!



---

# Category: Version 1.1.3

All the aspects of the API that were added, or had a new signature added, in the corresponding version of jQuery.

[jQuery 1.1.3 Release Notes](#)

Also in: [Events](#) > [Event Object](#)

---

## event.which

For key or mouse events, this property indicates the specific key or button that was pressed.

Also in: [Properties](#) > [Properties of the Global jQuery Object](#) | [Utilities](#)

---

## jQuery.browser

Contains flags for the useragent, read from navigator.userAgent. We recommend against using this property; please try to use feature detection instead (see jQuery.support). jQuery.browser may be moved to a plugin in a future release of jQuery.

Also in: [Utilities](#)

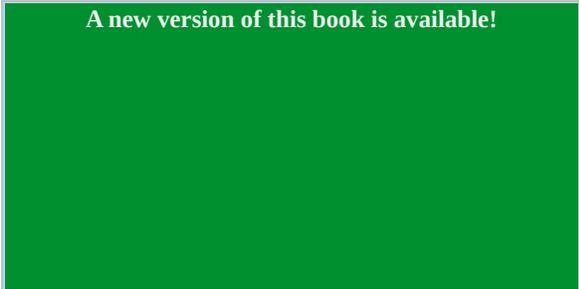
---

## jQuery.unique()

Sorts an array of DOM elements, in place, with the duplicates removed. Note that this only works on arrays of DOM elements, not strings or numbers.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

## Category: Version 1.1.4

All the aspects of the API that were added, or had a new signature added, in the corresponding version of jQuery.

[jQuery 1.1.4 Release Notes](#).

Also in: [Selectors > Content Filter](#)

---

## **:contains() Selector**

Select all elements that contain the specified text.

Also in: [Events](#) > [Event Object](#)

---

## event.relatedTarget

The other DOM element involved in the event, if any.

Also in: [Selectors](#) > [Child Filter](#)

---

## **:first-child Selector**

Selects all elements that are the first child of their parent.

Also in: [Selectors > Content Filter](#) | [Selectors > jQuery Extensions](#)

---

## **:has() Selector**

Selects elements which contain at least one element that matches the specified selector.

Also in: [Utilities](#)

---

## jQuery.isXMLDoc()

Check to see if a DOM node is within an XML document (or is an XML document).

Also in: [Selectors](#) > [Child Filter](#)

---

## **:last-child Selector**

Selects all elements that are the last child of their parent.

Also in: [Selectors](#) > [Child Filter](#)

---

## **:nth-child() Selector**

Selects all elements that are the nth-child of their parent.

Also in: [Selectors](#) > [Child Filter](#)

---

## **:only-child Selector**

Selects all elements that are the only child of their parent.

Also in: [Traversing](#) > [Filtering](#)

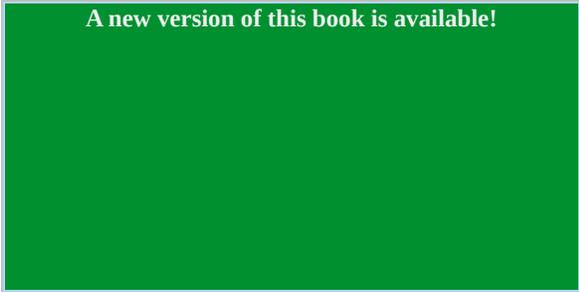
---

## **.slice()**

Reduce the set of matched elements to a subset specified by a range of indices.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

## Category: Version 1.2

All the aspects of the API that were added, or had a new signature added, in the corresponding version of jQuery.

[jQuery 1.2 Release Notes](#)

Also in: [Traversing](#) > [Miscellaneous Traversing](#)

---

## **.andSelf()**

Add the previous set of elements on the stack to the current set.

Also in: [Selectors > Basic Filter](#) | [Selectors > jQuery Extensions](#)

---

## **:animated Selector**

Select all elements that are in the progress of an animation at the time the selector is run.

Also in: [Traversing](#) > [Miscellaneous Traversing](#)

---

## `.contents()`

Get the children of each element in the set of matched elements, including text and comment nodes.

Also in: [Effects > Custom](#) | [Data](#) | [Utilities](#)

---

## **.dequeue()**

Execute the next function on the queue for the matched elements.

Also in: [Attributes](#) | [Manipulation > Class Attribute](#) | [CSS](#)

---

## **.hasClass()**

Determine whether any of the matched elements are assigned the given class.

Also in: [Selectors > Basic Filter](#) | [Selectors > jQuery Extensions](#)

---

## **:header Selector**

Selects all elements that are headers, like h1, h2, h3 and so on.

Also in: [Utilities](#)

---

## jQuery.inArray()

Search for a specified value within an array and return its index (or -1 if not found).

Also in: [Utilities](#)

---

## jQuery.isFunction()

Determine if the argument passed is a Javascript function object.

Also in: [Utilities](#)

---

## jQuery.makeArray()

Convert an array-like object into a true JavaScript array.

Also in: [Miscellaneous](#) > [Collection Manipulation](#) | [Forms](#) | [Ajax](#) > [Helper Functions](#)

---

## jQuery.param()

Create a serialized representation of an array or object, suitable for use in a URL query string or Ajax request.

Also in: [Traversing](#) > [Filtering](#)

---

## **.map()**

Pass each element in the current matched set through a function, producing a new jQuery object containing the return values.

Also in: [Traversing > Tree Traversal](#)

---

## **.nextAll()**

Get all following siblings of each element in the set of matched elements, optionally filtered by a selector.

Also in: [CSS](#) | [Offset](#) | [Manipulation](#) > [Style Properties](#)

---

## **.offset()**

Get the current coordinates of the first element, or set the coordinates of every element, in the set of matched elements, relative to the document.

Also in: [CSS](#) | [Offset](#) | [Manipulation](#) > [Style Properties](#)

---

## **.position()**

Get the current coordinates of the first element in the set of matched elements, relative to the offset parent.

Also in: [Traversing > Tree Traversal](#)

---

## **.prevAll()**

Get all preceding siblings of each element in the set of matched elements, optionally filtered by a selector.

Also in: [Effects > Custom](#) | [Data](#) | [Utilities](#)

---

## `.queue()`

Show or manipulate the queue of functions to be executed on the matched elements.

Also in: [Manipulation](#) > [DOM Replacement](#)

---

## **.replaceAll()**

Replace each target element with the set of matched elements.

Also in: [Manipulation](#) > [DOM Replacement](#)

---

## **.replaceWith()**

Replace each element in the set of matched elements with the provided new content and return the set of elements that was removed.

Also in: [Forms](#) | [Ajax](#) > [Helper Functions](#)

---

## **.serializeArray()**

Encode a set of form elements as an array of names and values.

Also in: [Effects](#) > [Custom](#)

---

## **.stop()**

Stop the currently-running animation on the matched elements.

Also in: [Events](#) > [Event Handler Attachment](#)

---

## **.triggerHandler()**

Execute all handlers attached to an element for an event.

Also in: [Manipulation](#) > [DOM Insertion, Around](#)

---

## **.wrapAll()**

Wrap an HTML structure around all elements in the set of matched elements.

Also in: [Manipulation](#) > [DOM Insertion, Around](#)

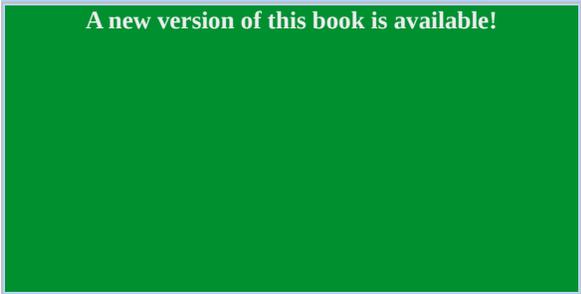
---

## **.wrapInner()**

Wrap an HTML structure around the content of each element in the set of matched elements.

POWERED BY **HERONOTE**

A new version of this book is available!



---

## Category: Version 1.2.3

All the aspects of the API that were added, or had a new signature added, in the corresponding version of jQuery.

Release Notes: [1.2.1](#), [1.2.2](#), [1.2.3](#).

Also in: [Data](#) | [Miscellaneous](#) > [Data Storage](#)

---

## **.data()**

Store arbitrary data associated with the matched elements or return the value at the named data store for the first element in the set of matched elements.

Also in: [Data](#) | [Utilities](#)

---

## jQuery.data()

Store arbitrary data associated with the specified element and/or return the value that was set.

Also in: [Data](#) | [Utilities](#)

---

## jQuery.removeData()

Remove a previously-stored piece of data.

Also in: [Data](#) | [Miscellaneous](#) > [Data Storage](#)

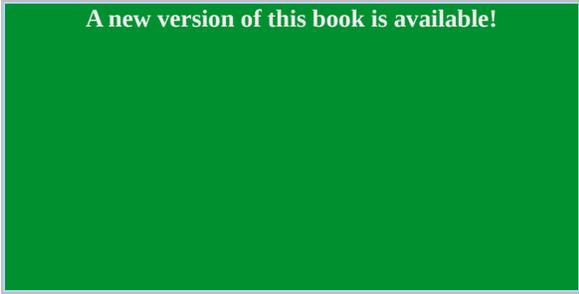
---

## `.removeData()`

Remove a previously-stored piece of data.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

## Category: Version 1.2.6

All the aspects of the API that were added, or had a new signature added, in the corresponding version of jQuery.

[jQuery 1.2.6 Release Notes](#).

Also in: [Events](#) > [Event Object](#)

---

## event.timeStamp

The difference in milliseconds between the time the browser created the event and January 1, 1970.

Also in: [CSS](#) | [Dimensions](#) | [Manipulation > Style Properties](#)

---

## **.innerHeight()**

Get the current computed height for the first element in the set of matched elements, including padding but not border.

Also in: [CSS](#) | [Dimensions](#) | [Manipulation > Style Properties](#)

---

## `.innerWidth()`

Get the current computed width for the first element in the set of matched elements, including padding but not border.

Also in: [CSS](#) | [Dimensions](#) | [Manipulation > Style Properties](#)

---

## **.outerHeight()**

Get the current computed height for the first element in the set of matched elements, including padding, border, and optionally margin. Returns an integer (without “px”) representation of the value or null if called on an empty set of elements.

Also in: [CSS](#) | [Dimensions](#) | [Manipulation](#) > [Style Properties](#)

---

## `.outerWidth()`

Get the current computed width for the first element in the set of matched elements, including padding and border.

Also in: [CSS](#) | [Offset](#) | [Manipulation](#) > [Style Properties](#)

---

## `.scrollLeft()`

Get the current horizontal position of the scroll bar for the first element in the set of matched elements or set the horizontal position of the scroll bar for every matched element.

Also in: [CSS](#) | [Offset](#) | [Manipulation](#) > [Style Properties](#)

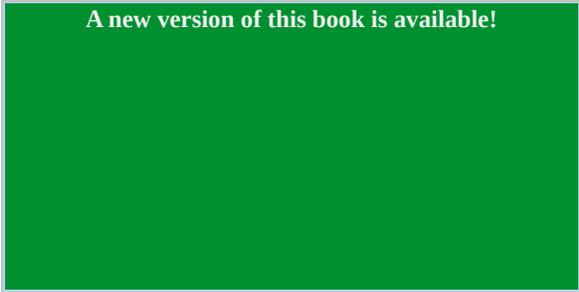
---

## **.scrollTop()**

Get the current vertical position of the scroll bar for the first element in the set of matched elements or set the vertical position of the scroll bar for every matched element.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

## Category: Version 1.3

All the aspects of the API that were added, or had a new signature added, in the corresponding version of jQuery.

Release Notes: [1.3](#), [1.3.1](#), [1.3.2](#)

Also in: [Traversing > Tree Traversal](#)

---

## **.closest()**

For each element in the set, get the first element that matches the selector by testing the element itself and traversing up through its ancestors in the DOM tree.

Also in: [Internals](#) | [Properties](#) > [Properties of jQuery Object Instances](#)

---

## **.context**

The DOM node context originally passed to `jQuery()`; if none was passed then context will likely be the document.

Also in: [Events](#) > [Event Handler Attachment](#)

---

## **.die()**

Remove event handlers previously attached using `.live()` from the elements.

Also in: [Events](#) > [Event Object](#)

---

## **event.currentTarget**

The current DOM element within the event bubbling phase.

Also in: [Events](#) > [Event Object](#)

---

## **event.isDefaultPrevented()**

Returns whether `event.preventDefault()` was ever called on this event object.

Also in: [Events](#) > [Event Object](#)

---

## **event.isImmediatePropagationStopped()**

Returns whether `event.stopImmediatePropagation()` was ever called on this event object.

Also in: [Events](#) > [Event Object](#)

---

## **event.isPropagationStopped()**

Returns whether `event.stopPropagation()` was ever called on this event object.

Also in: [Events](#) > [Event Object](#)

---

## **event.result**

The last value returned by an event handler that was triggered by this event, unless the value was undefined.

Also in: [Events](#) > [Event Object](#)

---

## **event.stopImmediatePropagation()**

Keeps the rest of the handlers from being executed and prevents the event from bubbling up the DOM tree.

Also in: [Data](#) | [Utilities](#)

---

## jQuery.dequeue()

Execute the next function on the queue for the matched element.

Also in: [Effects > Custom](#) | [Properties > Properties of the Global jQuery Object](#)

---

## jQuery.fx.off

Globally disable all animations.

Also in: [Utilities](#)

---

## jQuery.isArray()

Determine whether the argument is an array.

Also in: [Data](#) | [Utilities](#)

---

## jQuery.queue()

Show or manipulate the queue of functions to be executed on the matched element.

Also in: [Properties](#) > [Properties of the Global jQuery Object](#) | [Utilities](#)

---

## jQuery.support

A collection of properties that represent the presence of different browser features or bugs. Primarily intended for jQuery's internal use; specific properties may be removed when they are no longer needed internally to improve page startup performance.

Also in: [Events](#) > [Event Handler Attachment](#)

---

## **.live()**

Attach an event handler for all elements which match the current selector, now and in the future.

Also in: [Internals](#)

---

## **.pushStack()**

Add a collection of DOM elements onto the jQuery stack.

Also in: [Effects](#) > [Basics](#)

---

## **.toggle()**

Display or hide the matched elements.

Also in: [Attributes](#) | [Manipulation > Class Attribute](#) | [CSS](#)

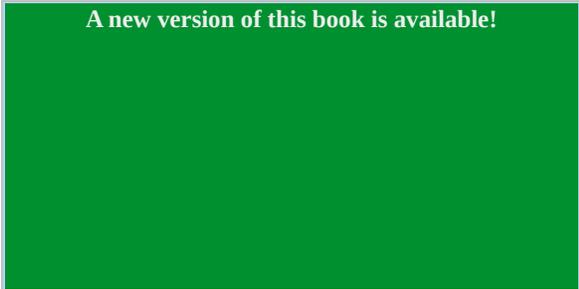
---

## **.toggleClass()**

Add or remove one or more classes from each element in the set of matched elements, depending on either the class's presence or the value of the switch argument.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

## Category: Version 1.4

All the aspects of the API that were added, or had a new signature added, in the corresponding version of jQuery.

[jQuery 1.4 Release Notes](#).

Also in: [Traversing](#) > [Miscellaneous Traversing](#)

---

## `.add()`

Add elements to the set of matched elements.

Also in: [Attributes](#) | [Manipulation > Class Attribute](#) | [CSS](#)

---

## **.addClass()**

Adds the specified class(es) to each of the set of matched elements.

Also in: [Manipulation > DOM Insertion, Outside](#)

---

## **.after()**

Insert content, specified by the parameter, after each element in the set of matched elements.

Also in: [Manipulation](#) > [DOM Insertion, Inside](#)

---

## **.append()**

Insert content, specified by the parameter, to the end of each element in the set of matched elements.

Also in: [Manipulation > DOM Insertion, Outside](#)

---

## **.before()**

Insert content, specified by the parameter, before each element in the set of matched elements.

Also in: [Events](#) > [Event Handler Attachment](#)

---

## **.bind()**

Attach a handler to an event for the elements.

Also in: [Effects > Custom](#) | [Data](#) | [Utilities](#)

---

## `.clearQueue()`

Remove from the queue all items that have not yet been run.

Also in: [Traversing](#) > [Tree Traversal](#)

---

## `.closest()`

For each element in the set, get the first element that matches the selector by testing the element itself and traversing up through its ancestors in the DOM tree.

Also in: [CSS](#) | [Manipulation](#) > [Style Properties](#)

---

## `.css()`

Get the value of a style property for the first element in the set of matched elements or set one or more CSS properties for every matched element.

Also in: [Data](#) | [Miscellaneous](#) > [Data Storage](#)

---

## **.data()**

Store arbitrary data associated with the matched elements or return the value at the named data store for the first element in the set of matched elements.

Also in: [Effects](#) > [Custom](#)

---

## `.delay()`

Set a timer to delay execution of subsequent items in the queue.

Also in: [Manipulation](#) > [DOM Removal](#)

---

## `.detach()`

Remove the set of matched elements from the DOM.

Also in: [Traversing](#) > [Filtering](#)

---

## **.filter()**

Reduce the set of matched elements to those that match the selector or pass the function's test.

Also in: [Traversing](#) > [Filtering](#)

---

## **.first()**

Reduce the set of matched elements to the first in the set.

Also in: [Events > Keyboard Events](#) | [Events > Mouse Events](#)

---

## **.focusin()**

Bind an event handler to the “focusin” event.

Also in: [Events > Keyboard Events](#) | [Events > Mouse Events](#)

---

## **.focusout()**

Bind an event handler to the “focusout” JavaScript event.

Also in: [Traversing](#) > [Filtering](#)

---

## `.has()`

Reduce the set of matched elements to those that have a descendant that matches the selector or DOM element.

Also in: [Attributes](#) | [Manipulation](#) > [DOM Insertion, Inside](#)

---

## `.html()`

Get the HTML contents of the first element in the set of matched elements or set the HTML contents of every matched element.

Also in: [Miscellaneous](#) > [DOM Element Methods](#)

---

## **.index()**

Search for a given element from among the matched elements.

Also in: [Core](#)

---

## jQuery()

Return a collection of matched elements either found in the DOM based on passed argument(s) or created by passing an HTML string.

Also in: [Utilities](#)

---

## jQuery.contains()

Check to see if a DOM element is a descendant of another DOM element.

Also in: [Data](#) | [Utilities](#)

---

## jQuery.data()

Store arbitrary data associated with the specified element and/or return the value that was set.

Also in: [Utilities](#)

---

## jQuery.isEmptyObject()

Check to see if an object is empty (contains no enumerable properties).

Also in: [Utilities](#)

---

## jQuery.isPlainObject()

Check to see if an object is a plain object (created using “{}” or “new Object”).

Also in: [Utilities](#)

---

## jQuery.noop()

An empty function.

Also in: [Miscellaneous](#) > [Collection Manipulation](#) | [Forms](#) | [Ajax](#) > [Helper Functions](#)

---

## jQuery.param()

Create a serialized representation of an array or object, suitable for use in a URL query string or Ajax request.

Also in: [Events](#) > [Event Handler Attachment](#) | [Utilities](#)

---

## jQuery.proxy()

Takes a function and returns a new one that will always have a particular context.

Also in: [Traversing](#) > [Filtering](#)

---

## `.last()`

Reduce the set of matched elements to the final one in the set.

Also in: [Traversing](#) > [Tree Traversal](#)

---

## **.nextUntil()**

Get all following siblings of each element up to but not including the element matched by the selector, DOM node, or jQuery object passed.

Also in: [Traversing > Filtering](#) | [Traversing > Miscellaneous Traversing](#)

---

## **.not()**

Remove elements from the set of matched elements.

Also in: [CSS](#) | [Offset](#) | [Manipulation](#) > [Style Properties](#)

---

## **.offset()**

Get the current coordinates of the first element, or set the coordinates of every element, in the set of matched elements, relative to the document.

Also in: [Traversing](#) > [Tree Traversal](#)

---

## **.parentsUntil()**

Get the ancestors of each element in the current set of matched elements, up to but not including the element matched by the selector, DOM node, or jQuery object.

Also in: [Manipulation](#) > [DOM Insertion, Inside](#)

---

## **.prepend()**

Insert content, specified by the parameter, to the beginning of each element in the set of matched elements.

Also in: [Traversing > Tree Traversal](#)

---

## `.prevUntil()`

Get all preceding siblings of each element up to but not including the element matched by the selector, DOM node, or jQuery object.

Also in: [Attributes](#) | [Manipulation](#) > [General Attributes](#)

---

## `.removeAttr()`

Remove an attribute from each element in the set of matched elements.

Also in: [Attributes](#) | [Manipulation > Class Attribute](#) | [CSS](#)

---

## **.removeClass()**

Remove a single class, multiple classes, or all classes from each element in the set of matched elements.

Also in: [Manipulation](#) > [DOM Replacement](#)

---

## **.replaceWith()**

Replace each element in the set of matched elements with the provided new content and return the set of elements that was removed.

Also in: [Manipulation](#) > [DOM Insertion, Inside](#)

---

## **.text()**

Get the combined text contents of each element in the set of matched elements, including their descendants, or set the text contents of the matched elements.

Also in: [Miscellaneous](#) > [DOM Element Methods](#)

---

## **.toArray()**

Retrieve all the DOM elements contained in the jQuery set, as an array.

Also in: [Attributes](#) | [Manipulation > Class Attribute](#) | [CSS](#)

---

## **.toggleClass()**

Add or remove one or more classes from each element in the set of matched elements, depending on either the class's presence or the value of the switch argument.

Also in: [Manipulation > DOM Insertion, Around](#) | [Manipulation > DOM Removal](#)

---

## `.unwrap()`

Remove the parents of the set of matched elements from the DOM, leaving the matched elements in their place.

Also in: [Attributes](#) | [Forms](#) | [Manipulation](#) > [General Attributes](#)

---

## `.val()`

Get the current value of the first element in the set of matched elements or set the value of every matched element.

Also in: [Manipulation](#) > [DOM Insertion, Around](#)

---

## `.wrap()`

Wrap an HTML structure around each element in the set of matched elements.

Also in: [Manipulation](#) > [DOM Insertion, Around](#)

---

## **.wrapAll()**

Wrap an HTML structure around all elements in the set of matched elements.

Also in: [Manipulation](#) > [DOM Insertion, Around](#)

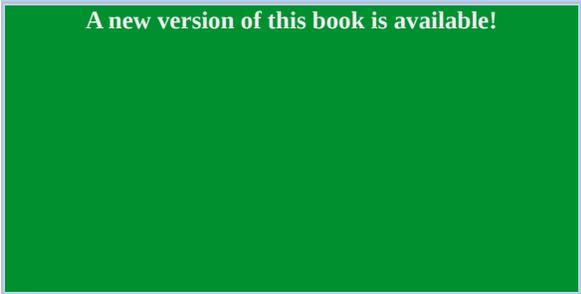
---

## **.wrapInner()**

Wrap an HTML structure around the content of each element in the set of matched elements.

POWERED BY **HERONOTE**

A new version of this book is available!



---

# Category: Version 1.4.1

All the aspects of the API that were added, or had a new signature added, in the corresponding version of jQuery.

[jQuery 1.4.1 Release Notes](#).

Also in: [Events > Event Handler Attachment](#)

---

## `.die()`

Remove event handlers previously attached using `.live()` from the elements.

Also in: [CSS](#) | [Dimensions](#) | [Manipulation > Style Properties](#)

---

## **.height()**

Get the current computed height for the first element in the set of matched elements or set the height of every matched element.

Also in: [Internals](#)

---

## jQuery.error()

Takes a string and throws an exception containing it.

Also in: [Utilities](#)

---

## jQuery.parseJSON()

Takes a well-formed JSON string and returns the resulting JavaScript object.

Also in: [CSS](#) | [Dimensions](#) | [Manipulation > Style Properties](#)

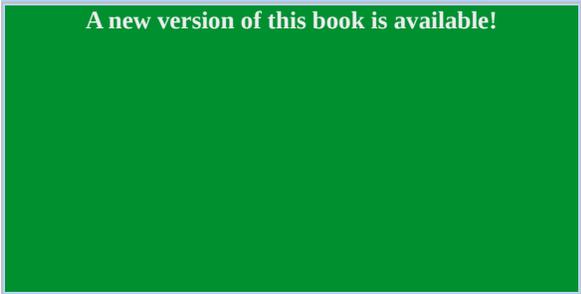
---

## `.width()`

Get the current computed width for the first element in the set of matched elements or set the width of every matched element.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

## Category: Version 1.4.2

All the aspects of the API that were added, or had a new signature added, in the corresponding version of jQuery.

[jQuery 1.4.2 Release Notes](#).

Also in: [Events > Event Handler Attachment](#)

---

## **.delegate()**

Attach a handler to one or more events for all elements that match the selector, now or in the future, based on a specific set of root elements.

Also in: [Events](#) > [Event Handler Attachment](#)

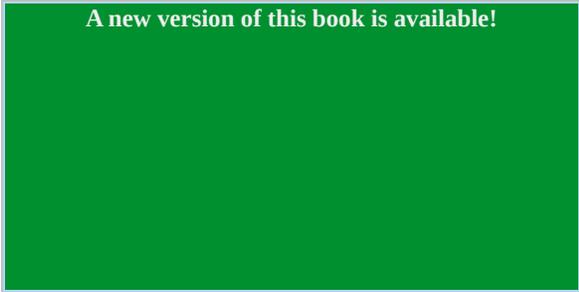
---

## **.undelegate()**

Remove a handler from the event for all elements which match the current selector, based upon a specific set of root elements.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

## Category: Version 1.4.3

All the aspects of the API that were added, or had a new signature added, in the corresponding version of jQuery.

[jQuery 1.4.3 Release Notes](#).

Also in: [Events > Event Handler Attachment](#)

---

## **.bind()**

Attach a handler to an event for the elements.

Also in: [Events > Form Events](#) | [Forms](#)

---

## **.blur()**

Bind an event handler to the “blur” JavaScript event, or trigger that event on an element.

Also in: [Events](#) > [Form Events](#) | [Forms](#)

---

## `.change()`

Bind an event handler to the “change” JavaScript event, or trigger that event on an element.

Also in: [Events](#) > [Mouse Events](#)

---

## **.click()**

Bind an event handler to the “click” JavaScript event, or trigger that event on an element.

Also in: [Data](#) | [Miscellaneous](#) > [Data Storage](#)

---

## **.data()**

Store arbitrary data associated with the matched elements or return the value at the named data store for the first element in the set of matched elements.

Also in: [Events](#) > [Mouse Events](#)

---

## **.dblclick()**

Bind an event handler to the “dblclick” JavaScript event, or trigger that event on an element.

Also in: [Events](#) > [Event Handler Attachment](#)

---

## **.delegate()**

Attach a handler to one or more events for all elements that match the selector, now or in the future, based on a specific set of root elements.

Also in: [Events](#) > [Event Handler Attachment](#)

---

## **.die()**

Remove event handlers previously attached using `.live()` from the elements.

Also in: [Events > Browser Events](#)

---

## **.error()**

Bind an event handler to the “error” JavaScript event.

Also in: [Events](#) > [Event Object](#)

---

## event.namespace

The namespace specified when the event was triggered.

Also in: [Effects](#) > [Fading](#)

---

## **.fadeIn()**

Display the matched elements by fading them to opaque.

Also in: [Effects > Fading](#)

---

## **.fadeOut()**

Hide the matched elements by fading them to transparent.

Also in: [Effects](#) > [Fading](#)

---

## **.fadeTo()**

Adjust the opacity of the matched elements.

Also in: [Events > Form Events](#) | [Forms](#)

---

## **.focus()**

Bind an event handler to the “focus” JavaScript event, or trigger that event on an element.

Also in: [Events > Keyboard Events](#) | [Events > Mouse Events](#)

---

## **.focusin()**

Bind an event handler to the “focusin” event.

Also in: [Events > Keyboard Events](#) | [Events > Mouse Events](#)

---

## **.focusout()**

Bind an event handler to the “focusout” JavaScript event.

Also in: [Effects](#) > [Basics](#)

---

## **.hide()**

Hide the matched elements.

Also in: [CSS](#)

---

## jQuery.cssHooks

Hook directly into jQuery to override how particular CSS properties are retrieved or set, normalize CSS property naming, or create custom properties.

Also in: [Data](#) | [Utilities](#)

---

## jQuery.data()

Store arbitrary data associated with the specified element and/or return the value that was set.

Also in: [Effects > Custom](#) | [Properties > Properties of the Global jQuery Object](#)

---

## jQuery.fx.interval

The rate (in milliseconds) at which animations fire.

Also in: [Utilities](#)

---

## jQuery.isWindow()

Determine whether the argument is a window.

Also in: [Utilities](#)

---

## jQuery.now()

Return a number representing the current time.

Also in: [Utilities](#)

---

## jQuery.type()

Determine the internal JavaScript `[[Class]]` of an object.

Also in: [Events](#) > [Keyboard Events](#)

---

## **.keydown()**

Bind an event handler to the “keydown” JavaScript event, or trigger that event on an element.

Also in: [Events](#) > [Keyboard Events](#)

---

## **.keypress()**

Bind an event handler to the “keypress” JavaScript event, or trigger that event on an element.

Also in: [Events](#) > [Keyboard Events](#)

---

## **.keyup()**

Bind an event handler to the “keyup” JavaScript event, or trigger that event on an element.

Also in: [Events](#) > [Document Loading](#)

---

## **.load()**

Bind an event handler to the “load” JavaScript event.

Also in: [Events](#) > [Mouse Events](#)

---

## `.mousedown()`

Bind an event handler to the “mousedown” JavaScript event, or trigger that event on an element.

Also in: [Events](#) > [Mouse Events](#)

---

## **.mouseenter()**

Bind an event handler to be fired when the mouse enters an element, or trigger that handler on an element.

Also in: [Events](#) > [Mouse Events](#)

---

## **.mouseleave()**

Bind an event handler to be fired when the mouse leaves an element, or trigger that handler on an element.

Also in: [Events](#) > [Mouse Events](#)

---

## `.mousemove()`

Bind an event handler to the “mousemove” JavaScript event, or trigger that event on an element.

Also in: [Events](#) > [Mouse Events](#)

---

## **.mouseout()**

Bind an event handler to the “mouseout” JavaScript event, or trigger that event on an element.

Also in: [Events](#) > [Mouse Events](#)

---

## **.mouseover()**

Bind an event handler to the “mouseover” JavaScript event, or trigger that event on an element.

Also in: [Events](#) > [Mouse Events](#)

---

## `.mouseup()`

Bind an event handler to the “mouseup” JavaScript event, or trigger that event on an element.

Also in: [Events](#) > [Browser Events](#)

---

## **.resize()**

Bind an event handler to the “resize” JavaScript event, or trigger that event on an element.

Also in: [Events](#) > [Browser Events](#)

---

## **.scroll()**

Bind an event handler to the “scroll” JavaScript event, or trigger that event on an element.

Also in: [Events > Form Events](#) | [Forms](#)

---

## **.select()**

Bind an event handler to the “select” JavaScript event, or trigger that event on an element.

Also in: [Effects](#) > [Basics](#)

---

## `.show()`

Display the matched elements.

Also in: [Effects](#) > [Sliding](#)

---

## **.slideDown()**

Display the matched elements with a sliding motion.

Also in: [Effects](#) > [Sliding](#)

---

## **.slideToggle()**

Display or hide the matched elements with a sliding motion.

Also in: [Effects](#) > [Sliding](#)

---

## `.slideUp()`

Hide the matched elements with a sliding motion.

Also in: [Events > Form Events](#) | [Forms](#)

---

## **.submit()**

Bind an event handler to the “submit” JavaScript event, or trigger that event on an element.

Also in: [Effects](#) > [Basics](#)

---

## **.toggle()**

Display or hide the matched elements.

Also in: [Events](#) > [Event Handler Attachment](#)

---

## **.unbind()**

Remove a previously-attached event handler from the elements.

Also in: [Events](#) > [Event Handler Attachment](#)

---

## **.undelegate()**

Remove a handler from the event for all elements which match the current selector, based upon a specific set of root elements.

Also in: [Events](#) > [Document Loading](#)

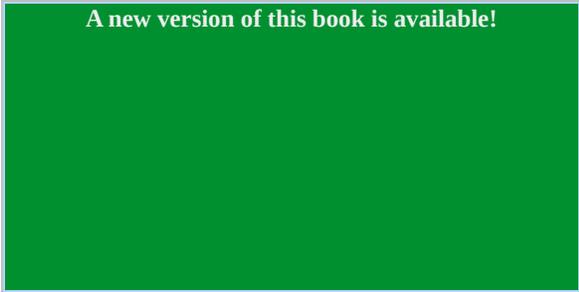
---

## **.unload()**

Bind an event handler to the “unload” JavaScript event.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

## Category: Version 1.4.4

All the aspects of the API that were added, or had a new signature added, in the corresponding version of jQuery. [jQuery 1.4.4 Release Notes](#).

Also in: [Effects](#) | [Effects > Fading](#)

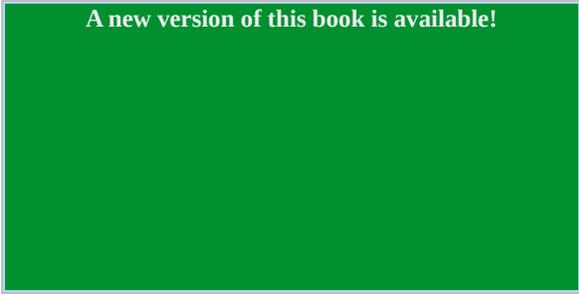
---

## `.fadeToggle()`

Display or hide the matched elements by animating their opacity.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

## Category: Version 1.5

All the aspects of the API that were added, or had a new signature added, in the corresponding version of jQuery.

jQuery 1.5 also includes a large rewrite of the Ajax module, which has a number of extensibility improvements. You can find out more about those improvements in the [Extending Ajax](#) documentation.

Additionally jQuery 1.5 includes a new Deferred callback management system you can learn more about in in the [Deferred Object](#) documentation.

Also in: [Manipulation](#) > [Copying](#)

---

## **.clone()**

Create a deep copy of the set of matched elements.

Also in: [Deferred Object](#)

---

## deferred.done()

Add handlers to be called when the Deferred object is resolved.

Also in: [Deferred Object](#)

---

## deferred.fail()

Add handlers to be called when the Deferred object is rejected.

Also in: [Deferred Object](#)

---

## deferred.isRejected()

Determine whether a Deferred object has been rejected.

Also in: [Deferred Object](#)

---

## deferred.isResolved()

Determine whether a Deferred object has been resolved.

Also in: [Deferred Object](#)

---

## deferred.promise()

Return a Deferred's Promise object.

Also in: [Deferred Object](#)

---

## deferred.reject()

Reject a Deferred object and call any failCallbacks with the given args.

Also in: [Deferred Object](#)

---

## deferred.rejectWith()

Reject a Deferred object and call any failCallbacks with the given context and args.

Also in: [Deferred Object](#)

---

## deferred.resolve()

Resolve a Deferred object and call any doneCallbacks with the given args.

Also in: [Deferred Object](#)

---

## deferred.resolveWith()

Resolve a Deferred object and call any doneCallbacks with the given context and args.

Also in: [Deferred Object](#)

---

## deferred.then()

Add handlers to be called when the Deferred object is resolved, rejected, or still in progress.

Also in: [Ajax > Low-Level Interface](#)

---

## jQuery.ajax()

Perform an asynchronous HTTP (Ajax) request.

Also in: [Ajax](#) > [Low-Level Interface](#)

---

## jQuery.ajaxPrefilter()

Handle custom Ajax options or modify existing options before each request is sent and before they are processed by \$.ajax().

Also in: [Ajax > Low-Level Interface](#)

---

## jQuery.ajaxTransport()

Creates an object that handles the actual transmission of Ajax data.

Also in: [Deferred Object](#)

---

## jQuery.Deferred()

A constructor function that returns a chainable utility object with methods to register multiple callbacks into callback queues, invoke callback queues, and relay the success or failure state of any synchronous or asynchronous function.

Also in: [Ajax](#) > [Shorthand Methods](#)

---

## jQuery.get()

Load data from the server using a HTTP GET request.

Also in: [Ajax](#) > [Shorthand Methods](#)

---

## jQuery.getJSON()

Load JSON-encoded data from the server using a GET HTTP request.

Also in: [Ajax > Shorthand Methods](#)

---

## jQuery.getScript()

Load a JavaScript file from the server using a GET HTTP request, then execute it.

Also in: [Data](#)

---

## jQuery.hasData()

Determine whether an element has any jQuery data associated with it.

Also in: [Utilities](#)

---

## jQuery.parseXML()

Parses a string into an XML document.

Also in: [Ajax](#) > [Shorthand Methods](#)

---

## jQuery.post()

Load data from the server using a HTTP POST request.

Also in: [Core](#)

---

## jQuery.sub()

Creates a new copy of jQuery whose properties and methods can be modified without affecting the original jQuery object.

Also in: [Core](#) | [Deferred Object](#)

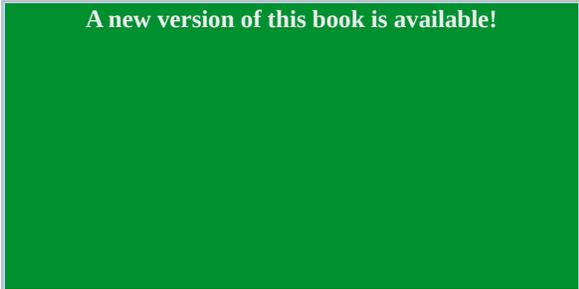
---

## jQuery.when()

Provides a way to execute callback functions based on one or more objects, usually Deferred objects that represent asynchronous events.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

# Category: Version 1.5.1

Aspects of the API that were changed in the corresponding version of jQuery. API changes in jQuery 1.5.1 dealt primarily with `jQuery.ajax` settings and `jQuery.support` properties.

Also in: [Ajax > Low-Level Interface](#)

---

## jQuery.ajax()

Perform an asynchronous HTTP (Ajax) request.

Also in: [Properties](#) > [Properties of the Global jQuery Object](#) | [Utilities](#)

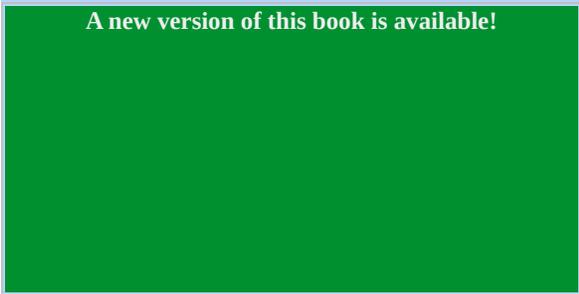
---

## jQuery.support

A collection of properties that represent the presence of different browser features or bugs. Primarily intended for jQuery's internal use; specific properties may be removed when they are no longer needed internally to improve page startup performance.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

## Category: Version 1.6

All the aspects of the API that were added, or had a new signature added, in the corresponding version of jQuery.

Also in: [Attributes](#) | [Manipulation](#) > [General Attributes](#)

---

## **.attr()**

Get the value of an attribute for the first element in the set of matched elements or set one or more attributes for every matched element.

Also in: [Traversing](#) > [Tree Traversal](#)

---

## **.closest()**

For each element in the set, get the first element that matches the selector by testing the element itself and traversing up through its ancestors in the DOM tree.

Also in: [Deferred Object](#)

---

## deferred.always()

Add handlers to be called when the Deferred object is either resolved or rejected.

Also in: [Deferred Object](#)

---

## deferred.pipe()

Utility method to filter and/or chain Deferreds.

Also in: [Traversing](#) > [Tree Traversal](#)

---

## **.find()**

Get the descendants of each element in the current set of matched elements, filtered by a selector, jQuery object, or element.

Also in: [Selectors > Basic Filter](#) | [Selectors > Form](#)

---

## **:focus Selector**

Selects element if it is currently focused.

Also in: [Traversing](#) > [Filtering](#)

---

## `.is()`

Check the current matched set of elements against a selector, element, or jQuery object and return true if at least one of these elements matches the given arguments.

Also in: [Core](#)

---

## jQuery.holdReady()

Holds or releases the execution of jQuery's ready event.

Also in: [Utilities](#)

---

## jQuery.map()

Translate all items in an array or object to new array of items.

Also in: [Events](#) > [Event Handler Attachment](#) | [Utilities](#)

---

## jQuery.proxy()

Takes a function and returns a new one that will always have a particular context.

Also in: [Traversing > Tree Traversal](#)

---

## **.nextUntil()**

Get all following siblings of each element up to but not including the element matched by the selector, DOM node, or jQuery object passed.

Also in: [Traversing](#) > [Tree Traversal](#)

---

## **.parentsUntil()**

Get the ancestors of each element in the current set of matched elements, up to but not including the element matched by the selector, DOM node, or jQuery object.

Also in: [Traversing](#) > [Tree Traversal](#)

---

## **.prevUntil()**

Get all preceding siblings of each element up to but not including the element matched by the selector, DOM node, or jQuery object.

Also in: [Deferred Object](#)

---

## **.promise()**

Return a Promise object to observe when all actions of a certain type bound to the collection, queued or not, have finished.

Also in: [Attributes](#) | [Manipulation](#) > [General Attributes](#)

---

## **.prop()**

Get the value of a property for the first element in the set of matched elements or set one or more properties for every matched element.

Also in: [Attributes](#) | [Manipulation](#) > [General Attributes](#)

---

## **.removeProp()**

Remove a property for the set of matched elements.

Also in: [Events](#) > [Event Handler Attachment](#)

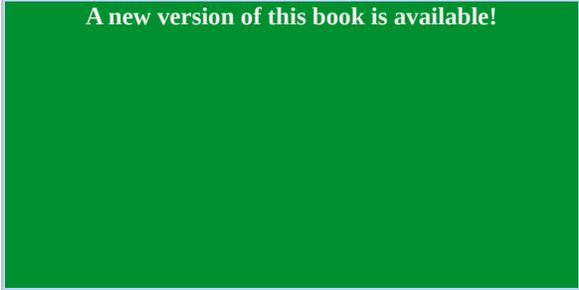
---

## **.undelegate()**

Remove a handler from the event for all elements which match the current selector, based upon a specific set of root elements.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

## Category: Version 1.7

Aspects of the API that were changed in the corresponding version of jQuery. API changes in jQuery 1.7.0 dealt primarily with the new Event APIs: `.on()` and `.off()`

Better Support for HTML5 in IE6/7/8

`jQuery.Callbacks()`

Toggle Animations Work Intuitively

For more information, see the Release Notes/Changelog at <http://blog.jquery.com/2011/11/03/jquery-1-7-released/>

Also in: `Callbacks Object`

---

## callbacks.add()

Add a callback or a collection of callbacks to a callback list.

Also in: [Callbacks Object](#)

---

## callbacks.disable()

Disable a callback list from doing anything more.

Also in: [Callbacks Object](#)

---

## callbacks.disabled()

Determine if the callbacks list has been disabled.

Also in: [Callbacks Object](#)

---

## callbacks.empty()

Remove all of the callbacks from a list.

Also in: [Callbacks Object](#)

---

## callbacks.fire()

Call all of the callbacks with the given arguments

Also in: [Callbacks Object](#)

---

## callbacks.fired()

Determine if the callbacks have already been called at least once.

Also in: [Callbacks Object](#)

---

## callbacks.fireWith()

Call all callbacks in a list with the given context and arguments.

Also in: [Callbacks Object](#)

---

## callbacks.has()

Determine whether a supplied callback is in a list

Also in: [Callbacks Object](#)

---

## callbacks.lock()

Lock a callback list in its current state.

Also in: [Callbacks Object](#)

---

## callbacks.locked()

Determine if the callbacks list has been locked.

Also in: [Callbacks Object](#)

---

## callbacks.remove()

Remove a callback or a collection of callbacks from a callback list.

Also in: [Deferred Object](#)

---

## deferred.notify()

Call the progressCallbacks on a Deferred object with the given args.

Also in: [Deferred Object](#)

---

## deferred.notifyWith()

Call the progressCallbacks on a Deferred object with the given context and args.

Also in: [Deferred Object](#)

---

## deferred.pipe()

Utility method to filter and/or chain Deferreds.

Also in: [Deferred Object](#)

---

## deferred.progress()

Add handlers to be called when the Deferred object generates progress notifications.

Also in: [Deferred Object](#)

---

## **deferred.state()**

Determine the current state of a Deferred object.

Also in: [Deferred Object](#)

---

## deferred.then()

Add handlers to be called when the Deferred object is resolved, rejected, or still in progress.

Also in: [Events](#) > [Event Object](#) | [Events](#)

---

## event.delegateTarget

The element where the currently-called jQuery event handler was attached.

Also in: [Traversing](#) > [Filtering](#)

---

## **.is()**

Check the current matched set of elements against a selector, element, or jQuery object and return true if at least one of these elements matches the given arguments.

Also in: [Callbacks Object](#)

---

## jQuery.Callbacks()

A multi-purpose callbacks list object that provides a powerful way to manage callback lists.

Also in: [Utilities](#)

---

## jQuery.isNumeric()

Determines whether its argument is a number.

Also in: [Events](#) > [Event Handler Attachment](#)

---

## **.off()**

Remove an event handler.

Also in: [Events](#) > [Event Handler Attachment](#)

---

## `.on()`

Attach an event handler function for one or more events to the selected elements.

Also in: [Attributes](#) | [Manipulation](#) > [General Attributes](#)

---

## **.removeAttr()**

Remove an attribute from each element in the set of matched elements.

Also in: [Data](#) | [Miscellaneous](#) > [Data Storage](#)

---

## **.removeData()**

Remove a previously-stored piece of data.

Also in: [Effects](#) > [Custom](#)

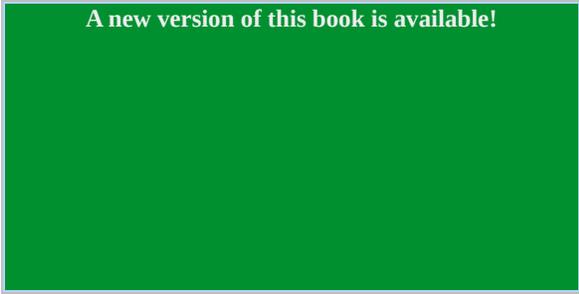
---

## **.stop()**

Stop the currently-running animation on the matched elements.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

## Category: Version 1.8

Aspects of the API that were changed in the corresponding version of jQuery. API changes in jQuery 1.8.0 dealt primarily with animations and the removal of some methods such as `deferred.isResolved()`, `deferred.isRejected()`, `$.curCSS()`, `$.attrFn()`, and `$(element).closest(Array)` returning Array.

For more information, see the Release Notes/Changelog at <http://blog.jquery.com/2012/08/09/jquery-1-8-released/>

Also in: [Traversing](#) > [Miscellaneous Traversing](#)

---

## **.addBack()**

Add the previous set of elements on the stack to the current set, optionally filtered by a selector.

Also in: [Deferred Object](#)

---

## deferred.pipe()

Utility method to filter and/or chain Deferreds.

Also in: [Selectors > Basic Filter](#) | [Selectors > jQuery Extensions](#)

---

## **:eq() Selector**

Select the element at index n within the matched set.

Also in: [Utilities](#)

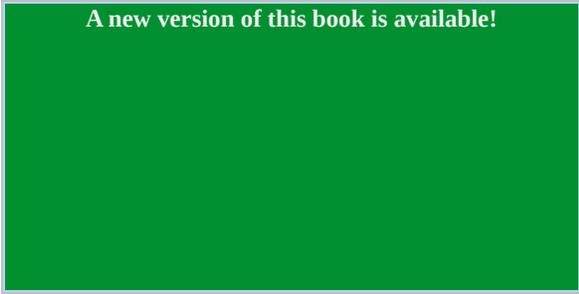
---

## jQuery.parseHTML()

Parses a string into an array of DOM nodes.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

## Category: Version 1.9

Aspects of the API that were changed in the corresponding version of jQuery. Changes in jQuery 1.9 dealt primarily removal or modification of several APIs that behaved inconsistently or inefficiently in the past. A [jQuery Migrate Plugin](#) was offered to help developers with a transitional upgrade path.

For more information, see the [jQuery Core 1.9 Upgrade guide](#) and the [Release Notes/Changelog](#)

Also in: [CSS](#) | [Manipulation > Style Properties](#)

---

## **.css()**

Get the value of a style property for the first element in the set of matched elements or set one or more CSS properties for every matched element.

Also in: [Effects](#) > [Custom](#)

---

## **.finish()**

Stop the currently-running animation, remove all queued animations, and complete all animations for the matched elements.

Also in: [Selectors](#) > [Child Filter](#)

---

## **:first-of-type Selector**

Selects all elements that are the first among siblings of the same element name.

Also in: [Selectors](#) > [Basic Filter](#)

---

## **:lang Selector**

Selects all elements of the specified language.

Also in: [Selectors](#) > [Child Filter](#)

---

## **:last-of-type Selector**

Selects all elements that are the last among siblings of the same element name.

Also in: [Selectors](#) > [Child Filter](#)

---

## **:nth-last-child() Selector**

Selects all elements that are the nth-child of their parent, counting from the last element to the first.

Also in: [Selectors](#) > [Child Filter](#)

---

## **:nth-last-of-type()** Selector

Selects all elements that are the nth-child of their parent, counting from the last element to the first.

Also in: [Selectors](#) > [Child Filter](#)

---

## **:nth-of-type() Selector**

Selects all elements that are the nth child of their parent in relation to siblings with the same element name.

Also in: [Selectors > Child Filter](#)

---

## **:only-of-type Selector**

Selects all elements that have no siblings with the same element name.

Also in: [Selectors](#) > [Basic Filter](#)

---

## **:root Selector**

Selects the element that is the root of the document.

Also in: [Selectors](#) > [Basic Filter](#)

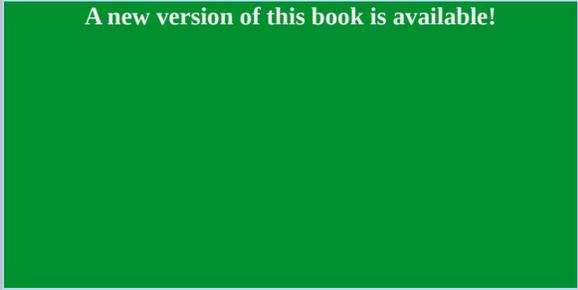
---

## **:target Selector**

Selects the target element indicated by the fragment identifier of the document's URI.

POWERED BY [HERONOTE](#)

A new version of this book is available!



# Types

JavaScript provides several built-in datatypes. In addition to those, this page documents virtual types like Selectors, enhanced pseudo-types like Events and all and everything you wanted to know about Functions.

You should be able to try out most of the examples below by just copying them to your browser's JavaScript Console (Chrome, Safari with Develop menu activated, IE 8+) or [Firebug](#) console ([Firefox](#)).

Whenever an example mentions that a type defaults to a boolean value, the result is good to know when using that type in a boolean context:

```
1 | var x = "";  
2 | if ( x ) {  
3 |     console.log( "x defaulted to true" );  
4 | } else {  
5 |     console.log( "x defaulted to false" );  
6 | }
```

In this case, "x defaulted to false" is printed.

To keep the examples short, the invert ("not") operator and double-negation are used to show a boolean context:

```
1 | var x = "";  
2 | !x // true  
3 | !!x // false (Double negation: Since "not (empt
```



On to the actual types.

---

# Contents

- 1 [String](#)
  - [Quoting](#)
  - [Built-in Methods](#)
  - [Length Property](#)
  - [Boolean Default](#)
  
- 2 [htmlString](#)
- 3 [Number](#)
  - [Boolean Default](#)
  - [Math](#)
  - [Parsing Numbers](#)
  - [Numbers to Strings](#)
  - [NaN and Infinity](#)
  - [Integer](#)
  - [Float](#)
  
- 4 [Boolean](#)
- 5 [Object](#)
  - [Dot Notation](#)
  - [Array Notation](#)
  - [Iteration](#)
  - [Boolean default](#)
  - [Prototype](#)
  
- 6 [Array](#)
  - [Iteration](#)
  - [Boolean Default](#)
  - [Array<Type> Notation](#)
  
- 7 [PlainObject](#)

- 8 [Function](#)
  - [Arguments](#)
  - [Context, Call and Apply](#)
  - [Scope](#)
  - [Closures](#)
  - [Proxy Pattern](#)
  
- 9 [Selector](#)
- 10 [Event](#)
- 11 [Element](#)
- 12 [jQuery](#)
- 13 [XMLHttpRequest](#)
- 14 [jqXHR](#)
- 15 [Deferred Object](#)
- 16 [Promise Object](#)
- 17 [Callbacks Object](#)

## String

A string in JavaScript is an immutable object that contains none, one or many characters.

```
1 | "I'm a String in JavaScript!"
2 | 'So am I!'
```

The type of a string is "string".

```
1 | typeof "some string"; // "string"
```

## Quoting

A string can be defined using single or double quotes. You can nest single quotes inside of double quotes, and the other way around. To mix double quotes with double quotes (or single with single), the nested ones have to be escaped with a backslash.

```
1 | "You make 'me' sad."
2 | 'That\'s "cranking" good fun!'
3 | "<a href=\"home\">Home</a>"
```

## Built-in Methods

A string in JavaScript has some built-in methods to manipulate the string, though the result is always a new string - or something else, eg. `split` returns an [array](#).

```
1 | "hello".charAt( 0 ) // "h"
2 | "hello".toUpperCase() // "HELLO"
3 | "Hello".toLowerCase() // "hello"
```

```
4 | "hello".replace( /e|o/g, "x" ) // "hxllx"  
5 | "1,2,3".split( ",", " " ) // [ "1", "2", "3" ]
```

## Length Property

All strings have a length property.

```
1 | "Hello".length // 5  
2 | "".length // 0
```

## Boolean Default

An empty string defaults to false:

```
1 | !"" // true  
2 | !!"" // false  
3 | !"hello" // false  
4 | !"true" // false  
5 | !new Boolean( false ) // false
```

## htmlString

A string is designated **htmlString** in jQuery documentation when it is used to represent one or more DOM elements, typically to be created and inserted in the document. When passed as an argument of the `jQuery()` function, the string is identified as HTML if it starts with `<tag ... >` and is parsed as such until the final `>` character.

For explicit parsing of a string to HTML, the `$.parseHTML()` method is available as of jQuery 1.8.

```
1 // Appends <b>hello</b>:  
2 $( "<b>hello</b>" ).appendTo( "body" );  
3  
4 // Appends <b>hello</b>:  
5 $( "<b>hello</b>bye" ).appendTo( "body" );  
6  
7 // Syntax error, unrecognized expression: bye<  
8 $( "bye<b>hello</b>" ).appendTo( "body" );  
9  
10 // Appends bye<b>hello</b>:  
11 $( $.parseHTML( "bye<b>hello</b>" ) ).appendTo  
12  
13 // Appends <b>hello</b>wait<b>bye</b>:  
14 $( "<b>hello</b>wait<b>bye</b>" ).appendTo( "b
```

## Number

Numbers in JavaScript are double-precision 64-bit format IEEE 754 values. They are immutable, just as [strings](#). All operators common in c-based languages are available to work with numbers (+, -, \*, /, %, =, +=, -=, \*=, /=, ++, --).

```
1 | 12
2 | 3.543
```

The type of a number is "number".

```
1 | typeof 12 // "number"
2 | typeof 3.543 // "number"
```

## Boolean Default

If a number is zero, it defaults to false:

```
1 | !0 // true
2 | !!0 // false
3 | !1 // false
4 | !-1 // false
```

Due to the implementation of numbers as double-precision values, the following result is not an error:

```
1 | 0.1 + 0.2 // 0.30000000000000004
```

## Math

JavaScript provides utilities to work with numbers in the Math object:

```
1 | Math.PI // 3.141592653589793
2 | Math.cos( Math.PI ) // -1
```

## Parsing Numbers

`parseInt` and `parseFloat` help parsing strings into numbers. Both do some implicit conversion if the base isn't specified:

```
1 | parseInt( "123" ) = 123 // (implicit decimal)
2 | parseInt( "010" ) = 8 // (implicit octal)
3 | parseInt( "0xCAFE" ) = 51966 // (implicit hexadec)
4 | parseInt( "010", 10 ) = 10 // (explicit decimal)
5 | parseInt( "11", 2 ) = 3 // (explicit binary)
6 | parseFloat( "10.10" ) = 10.1
```

## Numbers to Strings

When appending numbers to string, the result is always a string. The operator is the same, so be careful: If you want to add numbers and then append them to a string, put parentheses around them:

```
1 | "" + 1 + 2; // "12"
2 | "" + ( 1 + 2 ); // "3"
3 | "" + 0.0000001; // "1e-7"
4 | parseInt( 0.0000001 ); // 1 (!)
```

Or you use the `String` class provided by javascript, which try to parse a value as string:

```
1 | String( 1 ) + String( 2 ); // "12"
```

```
2 | String( 1 + 2 ); // "3"
```

## NaN and Infinity

Parsing something that isn't a number results in NaN. isNaN helps to detect those cases:

```
1 | parseInt( "hello", 10 ) // NaN
2 | isNaN( parseInt( "hello", 10 ) ) // true
```

Division by zero results in Infinity:

```
1 | 1 / 0 // Infinity
```

Both NaN and Infinity are of type "number":

```
1 | typeof NaN // "number"
2 | typeof Infinity // "number"
```

Note that NaN compares in a strange way:

```
1 | NaN == NaN // false (!)
```

But:

```
1 | Infinity == Infinity // true
```

## Integer

An integer is a plain Number type, but whenever explicitly mentioned,

indicates that a non-floating-point number is expected.

## **Float**

A float is a plain Number type, just as Integer, but whenever explicitly mentioned, indicates that a floating-point number is expected.

---

## Boolean

A boolean in JavaScript can be either true or false:

```
1 | if ( true ) console.log( "always!" );  
2 | if ( false ) console.log( "never!" );
```

## Object

Everything in JavaScript is an object, though some are more objective (haha). The easiest way to create an object is the object literal:

```
1 | var x = {};  
2 | var y = {  
3 |     name: "Pete",  
4 |     age: 15  
5 | };
```

The type of an object is "object":

```
1 | typeof {} // "object"
```

## Dot Notation

You can write and read properties of an object using the dot notation:

```
1 | y.name // "Pete"  
2 | y.age // 15  
3 | x.name = y.name + " Pan" // "Pete Pan"  
4 | x.age = y.age + 1 // 16
```

## Array Notation

Or you write and read properties using the array notation, which allows you to dynamically choose the property:

```
1 | var operations = {  
2 |     increase: "++",  
3 |     decrease: "--",  
4 | }
```

```
4 | };
5 | var operation = "increase";
6 | operations[ operation ] // "++"
7 | operations[ "multiply" ] = "*"; // "*"
```

## Iteration

Iterating over objects is easy with the for-in-loop:

```
1 | var obj = {
2 |   name: "Pete",
3 |   age: 15
4 | };
5 | for( key in obj ) {
6 |   alert( "key is " + [ key ] + ", value is " +
7 | }
```

Note that for-in-loop can be spoiled by extending Object.prototype (see [Object.prototype is verboten](#)) so take care when using other libraries.

jQuery provides a generic [each function](#) to iterate over properties of objects, as well as elements of arrays:

```
1 | jQuery.each( obj, function( key, value ) {
2 |   console.log( "key", key, "value", value );
3 | });
```

The drawback is that the callback is called in the context of each value and you therefore lose the context of your own object if applicable. More on this below at Functions.

## Boolean default

An object, no matter if it has properties or not, never defaults to false:

```
1 | !{} // false
2 | !!{} // true
```

## Prototype

All objects have a prototype property. Whenever the interpreter looks for a property, it also checks in the object's prototype if the property is not found on the object itself. jQuery uses the prototype extensively to add methods to jQuery instances. Internally, jQuery makes `jQuery.fn` an alias of `jQuery.prototype` so you can use either one (though plugin developers have standardized on `fn`).

```
1 | var form = $("#myForm");
2 | console.log( form.clearForm ); // undefined
3 |
4 | // jQuery.fn == jQuery.prototype
5 | jQuery.fn.clearForm = function() {
6 |     return this.find( ":input" ).each(function()
7 |         this.value = "";
8 |     }).end();
9 | };
10 |
11 | // works for all instances of jQuery objects,
12 | // the new method was added to the prototype
13 | console.log( form.clearForm ); // function
14 | form.clearForm();
```

## Array

Arrays in JavaScript are mutable lists with a few built-in methods. You can define arrays using the array literal:

```
1 | var x = [];  
2 | var y = [ 1, 2, 3 ];
```

The type of an array is "object":

```
1 | typeof []; // "object"  
2 | typeof [ 1, 2, 3 ]; // "object"
```

Reading and writing elements to an array uses the array-notation:

```
1 | x[ 0 ] = 1;  
2 | y[ 2 ] // 3
```

## Iteration

An array has a length property that is useful for iteration:

```
1 | for ( var i = 0; i < a.length; i++ ) {  
2 |     // Do something with a[i]  
3 | }
```

When performance is critical, reading the length property only once can help to speed things up. This should be used only when a performance bottleneck was discovered:

```
1 | for ( var i = 0, j = a.length; i < j; i++ ) {
```

```
2 | // Do something with a[i]
3 | }
```

Another variation defines a variable that is filled for each iteration, removing the array-notation from the loop-body. It does not work when the array contains 0 or empty strings!

```
1 | for ( var i = 0, item; item = a[i]; i++ ) {
2 | // Do something with item
3 | }
```

jQuery provides a generic [each function](#) to iterate over element of arrays, as well as properties of objects:

```
1 | var x = [ 1, 2, 3 ];
2 | jQuery.each( x, function( index, value ) {
3 |     console.log( "index", index, "value", value )
4 | });
```

The drawback is that the callback is called in the context of each value and you therefore lose the context of your own object if applicable. More on this below at Functions.

The length property can also be used to add elements to the end of an array. That is equivalent to using the push-method:

```
1 | var x = [];
2 | x.push( 1 );
3 | x[ x.length ] = 2;
4 | x // [ 1, 2 ]
```

You'll see both variations a lot when looking through JavaScript library code.

Other built-in methods are reverse, join, shift, unshift, pop, slice, splice and sort:

```
1 | var x = [ 0, 3, 1, 2 ];
2 | x.reverse() // [ 2, 1, 3, 0 ]
3 | x.join(" - ") // "2 - 1 - 3 - 0"
4 | x.pop() // [ 2, 1, 3 ]
5 | x.unshift( -1 ) // [ -1, 2, 1, 3 ]
6 | x.shift() // [ 2, 1, 3 ]
7 | x.sort() // [ 1, 2, 3 ]
8 | x.splice( 1, 2 ) // [ 2, 3 ]
```

Note: .unshift() method does not return a length property in Internet Explorer.

## Boolean Default

An array, no matter if it has elements or not, never defaults to false:

```
1 | ![] // false
2 | !![] // true
```

## Array<Type> Notation

In the jQuery API you'll often find the notation of Array<Type>:

```
dragPrevention Array<String>
```

This indicates that the method doesn't only expect an array as the argument, but also specifies the expected type. The notation is borrowed from Java 5's generics notation (or C++ templates).

## PlainObject

The PlainObject type is a JavaScript object containing zero or more key-value pairs. The plain object is, in other words, an `object` object. It is designated "plain" in jQuery documentation to distinguish it from other kinds of JavaScript objects: for example, `null`, user-defined arrays, and host objects such as `document`, all of which have a `typeof` value of "object." The `jQuery.isPlainObject()` method identifies whether the passed argument is a plain object or not, as demonstrated below:

```
1  var a = [];  
2  var d = document;  
3  var o = {};  
4  
5  typeof a; // object  
6  typeof d; // object  
7  typeof o; // object  
8  
9  jQuery.isPlainObject( a ); // false  
10 jQuery.isPlainObject( d ); // false  
11 jQuery.isPlainObject( o ); // true
```

## Function

A function in JavaScript can be either named or anonymous. Any function can be assigned to a variable or passed to a method, but passing member functions this way can cause them to be called in the context of another object (i.e. with a different "this" object).

```
1 | function named() {}  
2 | var handler = function() {}
```

You see a lot of anonymous functions in jQuery code:

```
1 | $( document ).ready(function() {});  
2 | $( "a" ).click(function() {});  
3 | $.ajax({  
4 |   url: "someurl.php",  
5 |   success: function() {}  
6 | });
```

The type of a function is "function".

## Arguments

Inside a function a special variable "arguments" is always available. It's similar to an array in that it has a length property, but it lacks the built-in methods of an array. The elements of the pseudo-array are the argument of the function call.

```
1 | function log( x ) {  
2 |   console.log( typeof x, arguments.length );  
3 | }  
4 | log(); // "undefined", 0  
5 | log( 1 ); // "number", 1  
6 | log( "1", "2", "3" ); // "string", 3
```

The arguments object also has a callee property, which refers to the function you're inside of. For instance:

```
1 | var awesome = function() { return arguments.callee; }
2 | awesome() == awesome // true
```

## Context, Call and Apply

In JavaScript, the variable "this" always refers to the current context. By default, "this" refers to the window object. Within a function this context can change, depending on how the function is called.

All event handlers in jQuery are called with the handling element as the context.

```
1 | $( document ).ready(function() {
2 |     // this refers to window.document
3 | });
4 | $( "a" ).click(function() {
5 |     // this refers to an anchor DOM element
6 | });
```

You can specify the context for a function call using the function-built-in methods call and apply. The difference between them is how they pass arguments. Call passes all arguments through as arguments to the function, while apply accepts an array as the arguments.

```
1 | function scope() {
2 |     console.log( this, arguments.length );
3 | }
4 | scope() // window, 0
5 | scope.call( "foobar", [ 1, 2 ] ); // "foobar", .
6 | scope.apply( "foobar", [ 1, 2 ] ); // "foobar",
```

## Scope

In JavaScript, all variables defined inside a function are only visible inside that function scope. Consider the following example:

```
1 // global
2 var x = 0;
3 (function() {
4     // private
5     var x = 1;
6     console.log( x ); // 1
7 })();
8 console.log( x ); // 0
```

It defines a variable *x* in the global scope, then defines an anonymous function and executes it immediately (the additional parentheses are required for immediate execution). Inside the function another variable *x* is defined with a different value. It is only visible within that function and doesn't overwrite the global variable.

## Closures

Closures are created whenever a variable that is defined outside the current scope is accessed from within some inner scope. In the following example, the variable *counter* is visible within the *create*, *increment*, and *print* functions, but not outside of them.

```
1 function create() {
2     var counter = 0;
3     return {
4         increment: function() {
5             counter++;
6         },
7         print: function() {
```

```
8     console.log( counter );
9     }
10    }
11   }
12   var c = create();
13   c.increment();
14   c.print(); // 1
```

The pattern allows you to create objects with methods that operate on data that isn't visible to the outside—the very basis of object-oriented programming.

## Proxy Pattern

Combining the above knowledge gives you as a JavaScript developer quite a lot of power. One way to combine that is to implement a proxy pattern in JavaScript, enabling the basics of aspect-oriented programming (AOP):

```
1  (function() {
2    // log all calls to setArray
3    var proxied = jQuery.fn.setArray;
4    jQuery.fn.setArray = function() {
5      console.log( this, arguments );
6      return proxied.apply( this, arguments );
7    };
8  })();
```

The above wraps its code in a function to hide the "proxied"-variable. It saves jQuery's setArray-method in a closure and overwrites it. The proxy then logs all calls to the method and delegates the call to the original. Using apply(this, arguments) guarantees that the caller won't be able to notice the difference between the original and the proxied method.

## Callback

A callback is a plain JavaScript function passed to some method as an argument or option. Some callbacks are just events, called to give the user a chance to react when a certain state is triggered. jQuery's event system uses such callbacks everywhere:

```
1 | $( "body" ).click(function( event ) {  
2 |     console.log( "clicked: " + event.target );  
3 | });
```

Most callbacks provide arguments and a context. In the event-handler example, the callback is called with one argument, an Event. The context is set to the handling element, in the above example, `document.body`.

Some callbacks are required to return something, others make that return value optional. To prevent a form submission, a submit event handler can return false:

```
1 | $( "#myform" ).submit(function() {  
2 |     return false;  
3 | });
```

Instead of always returning false, the callback could check fields of the form for validity, and return false only when the form is invalid.

## Selector

A selector is used in jQuery to select DOM elements from a DOM document. That document is, in most cases, the DOM document present in all browsers, but can also be an XML document received via AJAX.

The selectors are a composition of CSS and custom additions. All selectors available in jQuery are documented on the [Selectors API page](#).

There are a lot of plugins that leverage jQuery's selectors in other ways. The validation plugin accepts a selector to specify a dependency, whether an input is required or not:

```
1 | emailrules: {  
2 |   required: "#email:filled"  
3 | }
```

This would make a checkbox with name "emailrules" required only if the user entered an email address in the email field, selected via its id, filtered via a custom selector ":filled" that the validation plugin provides.

If Selector is specified as the type of an argument, it accepts everything that the jQuery constructor accepts, eg. Strings, Elements, Lists of Elements.

---

## Event

jQuery's event system normalizes the event object according to W3C standards. The event object is guaranteed to be passed to the event handler (no checks for `window.event` required). It normalizes the target, `relatedTarget`, `which`, `metaKey` and `pageX/Y` properties and provides both `stopPropagation()` and `preventDefault()` methods.

Those properties are all documented, and accompanied by examples, on the [Event object](#) page.

The standard events in the Document Object Model are: `blur`, `focus`, `load`, `resize`, `scroll`, `unload`, `beforeunload`, `click`, `dblclick`, `mousedown`, `mouseup`, `mousemove`, `mouseover`, `mouseout`, `mouseenter`, `mouseleave`, `change`, `select`, `submit`, `keydown`, `keypress`, and `keyup`.

Since the DOM event names have predefined meanings for some elements, using them for other purposes is not recommended. jQuery's event model can trigger an event by any name on an element, and it is propagated up the DOM tree to which that element belongs, if any.

---

## Element

An element in the Document Object Model (DOM) has attributes, text and children. It provides methods to traverse the parent and children and to get access to its attributes. Due to a lot of flaws in DOM API specifications and implementations, those methods are no fun to use. jQuery provides a wrapper around those elements to help interacting with the DOM. But often enough you will be working directly with DOM elements, or see methods that (also) accept DOM elements as arguments.

Whenever you use jQuery's each-method, the context of your callback is set to a DOM element. That is also the case for event handlers.

Some properties of DOM elements are quite consistent among browsers. Consider this example of a simple on-blur-validation:

```
1 | $( ":text" ).blur(function() {  
2 |     if( !this.value ) {  
3 |         alert( "Please enter some text!" );  
4 |     }  
5 | });
```

You could replace `this.value` with `$(this).val()` to access the value of the text input via jQuery, but in that case you don't gain anything.

---

## jQuery

A jQuery object contains a collection of Document Object Model (DOM) elements that have been created from an HTML string or selected from a document. Since jQuery methods often use CSS selectors to match elements from a document, the set of elements in a jQuery object is often called a set of "matched elements" or "selected elements".

The jQuery object itself behaves much like an array; it has a `length` property and the elements in the object can be accessed by their numeric indices `[0]` to `[length-1]`. Note that a jQuery object is not actually a Javascript Array object, so it does not have all the methods of a true Array object such as `join()`.

Most frequently, you will use the `jQuery()` function to create a jQuery object. `jQuery()` can also be accessed by its familiar single-character alias of `$()`, unless you have called `jQuery.noConflict()` to disable this option. Many jQuery methods return the jQuery object itself, so that method calls can be chained:

```
1 | $( "p" ).css( "color", "red" ).find( ".special"
```

Whenever you use a "destructive" jQuery method that potentially changes the set of elements in the jQuery object, such as `.filter()` or `.find()`, that method actually returns a new jQuery object with the resulting elements. To return to the previous jQuery object, you use the `.end()` method.

A jQuery object may be empty, containing no DOM elements. You can create an empty jQuery object with `$()` (that is, passing no arguments at all). A jQuery object may also be empty if a selector doesn't select any elements, or if a chained method filters out all the elements. It is not an error; any further methods called on that jQuery object simply have no effect since they have no elements to act upon. So, in this example if there are no bad entries on the page then no elements will be colored red:

```
|
```

```
1 | $( ".badEntry" ).css({ color: "red" });
```

---

## XMLHttpRequest

Some of jQuery's AJAX functions return the native XMLHttpRequest (XHR) object, or pass it as an argument to success/error/complete handlers, so that you can do additional processing or monitoring on the request. Note that AJAX functions only return or pass an XHR object when an XHR object is actually used in the request. For example, JSONP requests and cross-domain GET requests use a script element rather than an XHR object.

Although the XHR object is a standard, there are variations in its behavior on different browsers. Refer to the W3C site and browsers' documentation for more information:

[W3C standard](#)

[Apple \(Safari\)](#)

[Mozilla \(Firefox\)](#)

[Microsoft \(Internet Explorer\)](#)

[Opera](#)

Google does not appear to have an official page for their XHR documentation for Chrome. As of version 5, Chrome does not support the use of the file protocol for XHR requests.

---

## jqXHR

As of jQuery 1.5, the [\\$.ajax\(\)](#) method returns the jqXHR object, which is a superset of the XMLHttpRequest object. For more information, see the [jqXHR](#) section of the [\\$.ajax](#) entry

---

## Deferred Object

As of jQuery 1.5, the [Deferred](#) object provides a way to register multiple callbacks into self-managed callback queues, invoke callback queues as appropriate, and relay the success or failure state of any synchronous or asynchronous function.

---

---

## Promise Object

This object provides a subset of the methods of the [Deferred](#) object (`then`, `done`, `fail`, `always`, `pipe`, `isResolved`, and `isRejected`) to prevent users from changing the state of the Deferred.

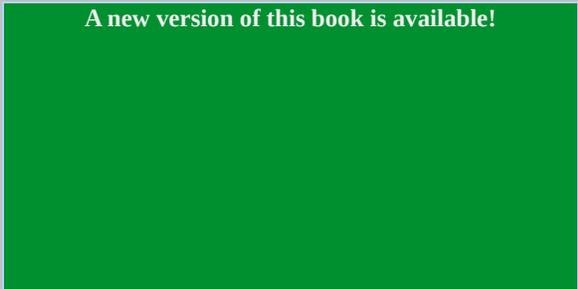
---

## Callbacks Object

A multi-purpose object that provides a powerful way to manage callback lists. It supports adding, removing, firing, and disabling callbacks. The Callbacks object is created and returned by the `$.Callbacks` function and subsequently returned by most of that function's methods.

POWERED BY [HERONOTE](#)

A new version of this book is available!



---

# jQuery UI 1.10 API Documentation

jQuery UI is a curated set of user interface interactions, effects, widgets, and themes built on top of the jQuery JavaScript Library. If you're new to jQuery UI, you might want to check out our [main site](#) for more information and full demos. If you're new to jQuery, you might also be interested in the [jQuery Learning Center](#) tutorials.

This site provides API documentation for jQuery UI 1.10. If you're working with jQuery UI 1.9, you can find the API documentation on [api.jqueryui.com/1.9/](http://api.jqueryui.com/1.9/). However, we would encourage you to upgrade to jQuery UI 1.10 in order to receive the best support and take advantage of recent bug fixes and enhancements. Check out the [release announcement](#) and [upgrade guide](#) to find out more about jQuery UI 1.10.

To get started, use the search at the top of the page, view the [full listing of entries](#), or browse by category from the sidebar.

jQuery UI 1.10 supports jQuery 1.6 and newer.

---

## Can't find what you're looking for?

Perhaps one of the following sites will help:

[jQuery UI 1.9 API Documentation](#)

[jQuery UI 1.8 API Documentation](#)

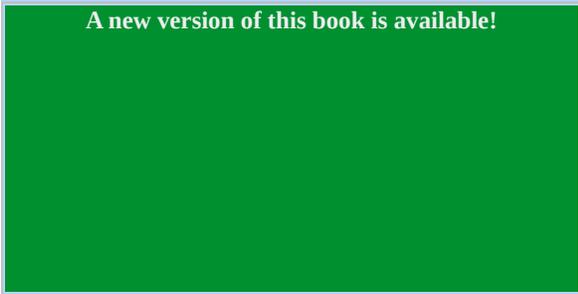
[jQuery Core API Documentation](#)

[jQuery Mobile API Documentation](#)

[jQuery Learning Center](#)

**POWERED BYHERONOTE**

**A new version of this book is available!**



---

# Category: All

Also in: [Widgets](#)

---

## Accordion Widget

Convert a pair of headers and content panels into an accordion.

Also in: [Effects](#) | [Effects Core](#) | [Method Overrides](#)

---

## **.addClass()**

Adds the specified class(es) to each of the set of matched elements while animating all style changes.

Also in: [Widgets](#)

---

## Autocomplete Widget

Autocomplete enables users to quickly find and select from a pre-populated list of values as they type, leveraging searching and filtering.

Also in: [Effects](#)

---

## Blind Effect

The blind effect hides or shows an element by wrapping the element in a container, and “pulling the blinds”

Also in: [Effects](#)

---

## Bounce Effect

The bounce effect bounces an element. When used with hide or show, the last or first bounce will also fade in/out.

Also in: [Widgets](#)

---

## Button Widget

Themable buttons and button sets.

Also in: [Effects](#)

---

## Clip Effect

The clip effect will hide or show an element by clipping the element vertically or horizontally.

Also in: [Selectors](#) | [UI Core](#)

---

## **:data() Selector**

Selects elements which have data stored under the specified key.

Also in: [Widgets](#)

---

## Datepicker Widget

Select a date from a popup or inline calendar

Also in: [Widgets](#)

---

## Dialog Widget

Open content in an interactive overlay.

Also in: [Methods](#) | [UI Core](#)

---

## **.disableSelection()**

Disable selection of text content within the set of matched elements.

Also in: [Interactions](#)

---

## Draggable Widget

Allow elements to be moved using the mouse.

Also in: [Effects](#)

---

## Drop Effect

The drop effect hides or shows an element fading in/out and sliding in a direction.

Also in: [Interactions](#)

---

## Draggable Widget

Create targets for draggable elements.

Also in: [Effects](#) | [Effects Core](#) | [Methods](#)

---

## **.effect()**

Apply an animation effect to an element.

Also in: [Methods](#) | [UI Core](#)

---

## **.enableSelection()**

Enable selection of text content within the set of matched elements.

Also in: [Effects](#)

---

## Explode Effect

The explode effect hides or shows an element by splitting it into pieces.

Also in: [Effects](#)

---

## Fade Effect

The fade effect hides or shows an element by fading it.

Also in: [Method Overrides](#) | [Methods](#) | [UI Core](#)

---

## **.focus()**

Asynchronously set focus to an element.

Also in: [Selectors](#) | [UI Core](#)

---

## **:focusable Selector**

Selects elements which can be focused.

Also in: [Effects](#)

---

## Fold Effect

The fold effect hides or shows an element by folding it.

Also in: [Effects](#) | [Effects Core](#) | [Method Overrides](#) | [Methods](#)

---

## **.hide()**

Hide the matched elements, using custom effects.

Also in: [Effects](#)

---

## Highlight Effect

The highlight effect hides or shows an element by animating its background color first.

Also in: [Utilities](#) | [Widgets](#)

---

## Widget Factory

Create stateful jQuery plugins using the same abstraction as all jQuery UI widgets.

Also in: [Widgets](#)

---

## Menu Widget

Themeable menu with mouse and keyboard interactions for navigation.

Also in: [Interactions](#) | [Utilities](#)

---

## Mouse Interaction

The base interaction layer.

Also in: [Method Overrides](#) | [Methods](#) | [Utilities](#)

---

## **.position()**

Position an element relative to another.

Also in: [Widgets](#)

---

## Progressbar Widget

Display status of a determinate or indeterminate process.

Also in: [Effects](#)

---

## Puff Effect

Creates a puff effect by scaling the element up and hiding it at the same time.

Also in: [Effects](#)

---

## Pulsate Effect

The pulsate effect hides or shows an element by pulsing it in or out.

Also in: [Effects](#) | [Effects Core](#) | [Method Overrides](#)

---

## **.removeClass()**

Removes the specified class(es) from each of the set of matched elements while animating all style changes.

Also in: [Methods](#) | [UI Core](#)

---

## **.removeUniqueId()**

Remove ids that were set by `.uniqueId()` for the set of matched elements.

Also in: [Interactions](#)

---

## Resizable Widget

Change the size of an element using the mouse.

Also in: [Effects](#)

---

## Scale Effect

Shrink or grow an element by a percentage factor.

Also in: [Methods](#) | [UI Core](#)

---

## **.scrollParent()**

Get the closest ancestor element that is scrollable.

Also in: [Interactions](#)

---

## Selectable Widget

Use the mouse to select elements, individually or in a group.

Also in: [Effects](#)

---

## Shake Effect

Shakes the element multiple times, vertically or horizontally.

Also in: [Effects](#) | [Effects Core](#) | [Method Overrides](#) | [Methods](#)

---

## **.show()**

Display the matched elements, using custom effects.

Also in: [Effects](#)

---

## Size Effect

Resize an element to a specified width and height.

Also in: [Effects](#)

---

## Slide Effect

Slides the element out of the viewport.

Also in: [Widgets](#)

---

## Slider Widget

Drag a handle to select a numeric value.

Also in: [Interactions](#)

---

## Sortable Widget

Reorder elements in a list or grid using the mouse.

Also in: [Widgets](#)

---

## Spinner Widget

Enhance a text input for entering numeric values, with up/down buttons and arrow key handling.

Also in: [Effects](#) | [Effects Core](#)

---

## **.switchClass()**

Adds and removes the specified class(es) to each of the set of matched elements while animating all style changes.

Also in: [Selectors](#) | [UI Core](#)

---

## **:tabbable Selector**

Selects elements which the user can focus via tabbing.

Also in: [Widgets](#)

---

## Tabs Widget

A single content area with multiple panels, each associated with a header in a list.

Also in: [Effects](#) | [Effects Core](#) | [Method Overrides](#) | [Methods](#)

---

## **.toggle()**

Display or hide the matched elements, using custom effects.

Also in: [Effects](#) | [Effects Core](#) | [Method Overrides](#)

---

## **.toggleClass()**

Add or remove one or more classes from each element in the set of matched elements, depending on either the class's presence or the value of the switch argument, while animating all style changes.

Also in: [Widgets](#)

---

## Tooltip Widget

Customizable, themeable tooltips, replacing native tooltips.

Also in: [Effects](#)

---

## Transfer Effect

Transfers the outline of an element to another element

Also in: [Methods](#) | [UI Core](#)

---

## **.uniqueId()**

Generate and apply a unique id for the set of matched elements.

Also in: [Methods](#) | [UI Core](#)

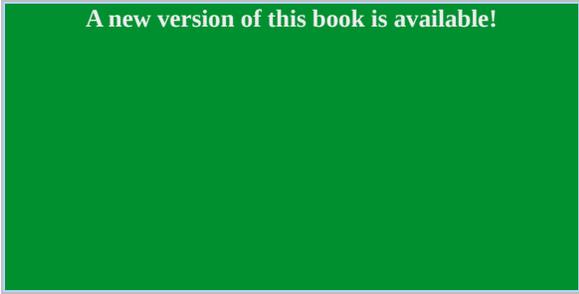
---

## **.zIndex()**

Get the z-index for an element.

**POWERED BYHERONOTE**

A new version of this book is available!



---

# Accordion Widget

Categories: [Widgets](#)

---

**Accordion Widget**

version added: 1.0

**Description:** *Convert a pair of headers and content panels into an accordion.*

## QuickNav

## Examples

### Options

[active](#)  
[animate](#)  
[collapsible](#)  
[disabled](#)  
[event](#)  
[header](#)  
[heightStyle](#)  
[icons](#)

### Methods

[destroy](#)  
[disable](#)  
[enable](#)  
[option](#)  
[refresh](#)  
[widget](#)

### Events

[activate](#)  
[beforeActivate](#)  
[create](#)

The markup of your accordion container needs pairs of headers and content panels:

```
1 <div id="accordion">
2   <h3>First header</h3>
3   <div>First content panel</div>
4   <h3>Second header</h3>
5   <div>Second content panel</div>
6 </div>
```

Accordions support arbitrary markup, but each content panel must always be the next sibling after its associated header. See the [header](#) option for information on how to use custom markup structures.

The panels can be activated programmatically by setting the [active](#) option.

## Keyboard interaction

When focus is on a header, the following key commands are available:

UP/LEFT - Move focus to the previous header. If on first header, moves focus to last header.

DOWN/RIGHT - Move focus to the next header. If on last header, moves focus to first header.

HOME - Move focus to the first header.

END - Move focus to the last header.

SPACE/ENTER - Activate panel associated with focused header.

When focus is in a panel:

CTRL+UP: Move focus to associated header.

## Dependencies

[UI Core](#)

[Widget Factory](#)

[Effects Core](#) (optional; for use with the `animate` option)

## Additional Notes:

This widget requires some functional CSS, otherwise it won't work. If you build a custom theme, use the widget's specific CSS file as a starting point.

## Options

### active

Type: [Boolean](#) or [Integer](#)

Default: `0`

Which panel is currently open.

#### Multiple types supported:

**Boolean:** Setting `active` to `false` will collapse all panels. This requires the `collapsible` option to be `true`.

**Integer:** The zero-based index of the panel that is active (open). A negative value selects panels going backward from the last panel.

#### Code examples:

Initialize the accordion with the active option specified:

```
1 | $( ".selector" ).accordion({ active: 2
```

Get or set the active option, after initialization:

```
1 | // getter
2 | var active = $( ".selector" ).accordion
3 |
4 | // setter
5 | $( ".selector" ).accordion( "option", "a
```

### animate

Default: **Type:** [Boolean](#) or [Number](#) or [String](#) or [Object](#)



If and how to animate changing panels.

### Multiple types supported:

**Boolean:** A value of `false` will disable animations.

**Number:** Duration in milliseconds with default easing.

**String:** Name of `easing` to use with default duration.

**Object:** Animation settings with `easing` and `duration` properties.

Can also contain a `down` property with any of the above options.

"Down" animations occur when the panel being activated has a lower index than the currently active panel.

### Code examples:

Initialize the accordion with the animate option specified:

```
1 | $( ".selector" ).accordion({ animate: "bounceslide"
◀ |
▶
```

Get or set the animate option, after initialization:

```
1 | // getter
2 | var animate = $( ".selector" ).accordion( "option"
3 |
4 | // setter
5 | $( ".selector" ).accordion( "option", "animate"
◀ |
▶
```

## collapsible

Type: `Boolean`

Default: `false`

Whether all the sections can be closed at once. Allows collapsing the active section.

### Code examples:

Initialize the accordion with the collapsible option specified:

```
1 | $( ".selector" ).accordion({ collapsible: true
```

Get or set the collapsible option, after initialization:

```
1 | // getter
2 | var collapsible = $( ".selector" ).accordion(
3 |
4 | // setter
5 | $( ".selector" ).accordion( "option", "collapsible"
```

## disabled

Type: **Boolean**

Default: **false**

Disables the accordion if set to **true**.

### Code examples:

Initialize the accordion with the disabled option specified:

```
1 | $( ".selector" ).accordion({ disabled: true });
```

Get or set the disabled option, after initialization:

```
1 | // getter
2 | var disabled = $( ".selector" ).accordion( "option"
3 |
4 | // setter
5 | $( ".selector" ).accordion( "option", "disabled"
```

## event

Type: [String](#)

Default: `"click"`

The event that accordion headers will react to in order to activate the associated panel. Multiple events can be specified, separated by a space.

### Code examples:

Initialize the accordion with the event option specified:

```
1 | $( ".selector" ).accordion({ event: "mouseover"
◀ | _____ ▶
```

Get or set the event option, after initialization:

```
1 | // getter
2 | var event = $( ".selector" ).accordion( "option"
3 |
4 | // setter
5 | $( ".selector" ).accordion( "option", "event"
◀ | _____ ▶
```

## header

Type: [Selector](#)

Default: `"> li > :first-child,> :not(li):even"`

Selector for the header element, applied via `.find()` on the main accordion element. Content panels must be the sibling immediately after their associated headers.

### Code examples:

Initialize the accordion with the header option specified:

```
1 | $( ".selector" ).accordion({ header: "h3" });
◀ | _____ ▶
```

Get or set the header option, after initialization:

```
1 // getter
2 var header = $( ".selector" ).accordion( "option"
3
4 // setter
5 $( ".selector" ).accordion( "option", "header"
```

## heightStyle

Type: [String](#)

Default: `"auto"`

Controls the height of the accordion and each panel.

Possible values:

`"auto"`: All panels will be set to the height of the tallest panel.

`"fill"`: Expand to the available height based on the accordion's parent height.

`"content"`: Each panel will be only as tall as its content.

### Code examples:

Initialize the accordion with the heightStyle option specified:

```
1 $( ".selector" ).accordion({ heightStyle: "fill"
```

Get or set the heightStyle option, after initialization:

```
1 // getter
2 var heightStyle = $( ".selector" ).accordion(
3
```

```
4 // setter
5 $( ".selector" ).accordion( "option", "heightStyle"
```

## icons

Type: [Object](#)

**Default:** `{ "header": "ui-icon-triangle-1-e", "activeHeader": "ui-icon-triangle-1-s" }`

Icons to use for headers, matching an icon defined by the jQuery UI CSS Framework. Set to `false` to have no icons displayed.

header (string, default: "ui-icon-triangle-1-e")

activeHeader (string, default: "ui-icon-triangle-1-s")

### Code examples:

Initialize the accordion with the icons option specified:

```
1 | $( ".selector" ).accordion({ icons: { "header"
```

Get or set the icons option, after initialization:

```
1 // getter
2 var icons = $( ".selector" ).accordion( "option"
3
4 // setter
5 $( ".selector" ).accordion( "option", "icons"
```

## Methods

### destroy()

Removes the accordion functionality completely. This will return the element back to its pre-init state.

This method does not accept any arguments.

#### Code examples:

Invoke the destroy method:

```
1 | $( ".selector" ).accordion( "destroy" );
```

### disable()

Disables the accordion.

This method does not accept any arguments.

#### Code examples:

Invoke the disable method:

```
1 | $( ".selector" ).accordion( "disable" );
```

### enable()

Enables the accordion.

This method does not accept any arguments.

### Code examples:

Invoke the enable method:

```
1 | $( ".selector" ).accordion( "enable" );
```

## option( optionName )

Returns: *Object*

Gets the value currently associated with the specified `optionName`.

**optionName**

Type: *String*

The name of the option to get.

### Code examples:

Invoke the method:

```
1 | var isDisabled = $( ".selector" ).accordion(
```

## option()

Returns: *PlainObject*

Gets an object containing key/value pairs representing the current accordion options hash.

This method does not accept any arguments.

### Code examples:

Invoke the method:

```
1 | var options = $( ".selector" ).accordion( "option"
```

## option( optionName, value )

Sets the value of the accordion option associated with the specified `optionName`.

### **optionName**

Type: String

The name of the option to set.

### **value**

Type: Object

A value to set for the option.

### **Code examples:**

Invoke the method:

```
1 | $( ".selector" ).accordion( "option", "disabled"
```

## option( options )

Sets one or more options for the accordion.

### **options**

Type: Object

A map of option-value pairs to set.

### **Code examples:**

Invoke the method:

```
1 | $( ".selector" ).accordion( "option", { disabled:
```

## refresh()

Recompute the height of the accordion panels. Results depend on the content and the `heightStyle` option.

This method does not accept any arguments.

### Code examples:

Invoke the refresh method:

```
1 | $( ".selector" ).accordion( "refresh" );
```

## widget()

Returns: *jQuery*

Returns a `jQuery` object containing the accordion.

This method does not accept any arguments.

### Code examples:

Invoke the widget method:

```
1 | var widget = $( ".selector" ).accordion( "widget" );
```

## Events

**activate( event, ui )**      *Type:* `accordionactivate`

Triggered after a panel has been activated (after animation completes). If the accordion was previously collapsed, `ui.oldHeader` and `ui.oldPanel` will be empty jQuery objects. If the accordion is collapsing, `ui.newHeader` and `ui.newPanel` will be empty jQuery objects.

**event**

Type: [Event](#)

**ui**

Type: [Object](#)

**newHeader**

Type: [jQuery](#)

The header that was just activated.

**oldHeader**

Type: [jQuery](#)

The header that was just deactivated.

**newPanel**

Type: [jQuery](#)

The panel that was just activated.

**oldPanel**

Type: [jQuery](#)

The panel that was just deactivated.

**Code examples:**

Initialize the accordion with the activate callback specified:

```
1 | $( ".selector" ).accordion({
```

```
2 | activate: function( event, ui ) {}  
3 | });
```

Bind an event listener to the accordionactivate event:

```
1 | $( ".selector" ).on( "accordionactivate", function
```

## beforeActivate( event, ui )

Type: `accordionbeforeactivate`

Triggered directly before a panel is activated. Can be canceled to prevent the panel from activating. If the accordion is currently collapsed, `ui.oldHeader` and `ui.oldPanel` will be empty jQuery objects. If the accordion is collapsing, `ui.newHeader` and `ui.newPanel` will be empty jQuery objects.

### event

Type: [Event](#)

### ui

Type: [Object](#)

#### newHeader

Type: [jQuery](#)

The header that is about to be activated.

#### oldHeader

Type: [jQuery](#)

The header that is about to be deactivated.

#### newPanel

Type: [jQuery](#)

The panel that is about to be activated.

#### oldPanel

Type: [jQuery](#)

The panel that is about to be deactivated.

### Code examples:

Initialize the accordion with the `beforeActivate` callback specified:

```
1 | $( ".selector" ).accordion({
2 |   beforeActivate: function( event, ui ) {}
3 | });
```

Bind an event listener to the `accordionbeforeactivate` event:

```
1 | $( ".selector" ).on( "accordionbeforeactivate"
```

## `create( event, ui )`

Type: `accordioncreate`

Triggered when the accordion is created. If the accordion is collapsed, `ui.header` and `ui.panel` will be empty jQuery objects.

### **event**

Type: [Event](#)

### **ui**

Type: [Object](#)

#### **header**

Type: [jQuery](#)

The active header.

#### **panel**

Type: [jQuery](#)

The active panel.

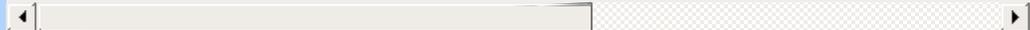
## Code examples:

Initialize the accordion with the create callback specified:

```
1 | $( ".selector" ).accordion({  
2 |   create: function( event, ui ) {}  
3 | });
```

Bind an event listener to the accordioncreate event:

```
1 | $( ".selector" ).on( "accordioncreate", function
```



## Example:

### *A simple jQuery UI Accordion*

```
1  <!doctype html>
2  <html lang="en">
3  <head>
4      <meta charset="utf-8">
5      <title>accordion demo</title>
6      <link rel="stylesheet" href="http://code.jq
7      <script src="http://code.jquery.com/jquery-:
8      <script src="http://code.jquery.com/ui/1.10
9  </head>
10 <body>
11
12 <div id="accordion">
13     <h3>Section 1</h3>
14     <div>
15         <p>Mauris mauris ante, blandit et, ultrico
16         Integer ut neque. Vivamus nisi metus, molo
17         condimentum sit amet, nunc. Nam a nibh. D
18         Nam mi. Proin viverra leo ut odio.</p>
19     </div>
20     <h3>Section 2</h3>
21     <div>
22         <p>Sed non urna. Phasellus eu ligula. Ves
23         Vivamus hendrerit, dolor aliquet laoreet,
24         faucibus interdum tellus libero ac justo.
25     </div>
26     <h3>Section 3</h3>
27     <div>
28         <p>Nam enim risus, molestie et, porta ac,
29         Quisque lobortis.Phasellus pellentesque p
30     <ul>
31         <li>List item one</li>
```

```
32     <li>List item two</li>
33     <li>List item three</li>
34   </ul>
35 </div>
36 </div>
37
38 <script>
39 $( "#accordion" ).accordion();
40 </script>
41
42 </body>
43 </html>
```

Demo

POWERED BYHERONOTE

A new version of this book is available!

---

# .addClass()

Categories: [Effects](#) | [Effects Core](#) | [Method Overrides](#)

```
.addClass( className [, duration ] [, easing ] [, complete ] )
```

Returns: [jQuery](#)

**Description:** Adds the specified class(es) to each of the set of matched elements while animating all style changes.

```
.addClass( className [, duration ] [, easing ] [, complete ] )
```

**className**

Type: [String](#)

One or more class names (space separated) to be added to the class attribute of each matched element.

**duration** (default: `400`)

Type: [Number](#) or [String](#)

A string or number determining how long the animation will run.

**easing** (default: `swing`)

Type: [String](#)

A string indicating which [easing](#) function to use for the transition.

**complete**

Type: [Function\(\)](#)

A function to call once the animation is complete.

Similar to native CSS transitions, jQuery UI's class animations provide a smooth transition from one state to another while allowing you to keep all the details about which styles to change in CSS and out of your JavaScript. All class animation methods, including `.addClass()`, support custom durations and easings, as well as providing a callback for when the animation completes.

Not all styles can be animated. For example, there is no way to animate a background image. Any styles that cannot be animated will be changed at the end of the animation.

This plugin extends jQuery's built-in `.addClass()` method. If jQuery

UI is not loaded, calling the `.addClass()` method may not fail directly, as the method still exists. However, the expected behavior will not occur.

## Example:

*Adds the class "big-blue" to the matched elements.*

```
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>addClass demo</title>
6   <link rel="stylesheet" href="http://code.jq
7   <style>
8   div {
9     width: 100px;
10    height: 100px;
11    background-color: #ccc;
12  }
13  .big-blue {
14    width: 200px;
15    height: 200px;
16    background-color: #00f;
17  }
18 </style>
19 <script src="http://code.jquery.com/jquery-:
20 <script src="http://code.jquery.com/ui/1.10
21 </head>
22 <body>
23
24 <div></div>
25
26 <script>
27 $( "div" ).click(function() {
28   $( this ).addClass( "big-blue", 1000, "ease
29 });
30 </script>
31
```

```
32 | </body>  
33 | </html>
```

Demo

**POWERED BYHERONOTE**

**A new version of this book is available!**

---

# Autocomplete Widget

Categories: [Widgets](#)

---

## Autocomplete Widget

version added: 1.8

**Description:** *Autocomplete enables users to quickly find and select from a pre-populated list of values as they type, leveraging searching and filtering.*

## QuickNav

## Examples

### Options

[appendTo](#)  
[autoFocus](#)  
[delay](#)  
[disabled](#)  
[minLength](#)  
[position](#)  
[source](#)

### Methods

[close](#)  
[destroy](#)  
[disable](#)  
[enable](#)  
[option](#)  
[search](#)  
[widget](#)

### Events

[change](#)  
[close](#)  
[create](#)  
[focus](#)  
[open](#)  
[response](#)  
[search](#)  
[select](#)

By giving an Autocomplete field focus or entering something into it, the plugin starts searching for entries that match and displays a list of values to choose from. By entering more characters, the user can filter down the list to better matches.

This can be used to choose previously selected values, such as entering tags for articles or entering email addresses from an address book. Autocomplete can also be used to populate associated information, such as entering a city name and getting the zip code.

You can pull data in from a local or remote source: Local is good for small data sets, e.g., an address book with 50 entries; remote is necessary for big data sets, such as a database with hundreds or millions of entries to select from. To find out more about customizing the data source, see the documentation for the `source` option.

## Keyboard interaction

When the menu is open, the following key commands are available:

UP - Move focus to the previous item. If on first item, move focus to the input. If on the input, move focus to last item.

DOWN - Move focus to the next item. If on last item, move

focus to the input. If on the input, move focus to the first item.

ESCAPE - Close the menu.

ENTER - Select the currently focused item and close the menu.

TAB - Select the currently focused item, close the menu, and move focus to the next focusable element.

PAGE UP/DOWN - Scroll through a page of items (based on height of menu). *It's generally a bad idea to display so many items that users need to page..*

When the menu is closed, the following key commands are available:

UP/DOWN - Open the menu, if the `minLength` has been met.

## Dependencies

[UI Core](#)

[Widget Factory](#)

[Position](#)

[Menu](#)

## Additional Notes:

This widget requires some functional CSS, otherwise it won't work. If you build a custom theme, use the widget's specific CSS file as a starting point.

This widget manipulates its element's value programmatically, therefore a native change may not be fired when the element's value changes.

## Options

### appendTo

Type: [Selector](#)

Default: `null`

Which element the menu should be appended to. When the value is `null`, the parents of the input field will be checked for a class of `ui-front`. If an element with the `ui-front` class is found, the menu will be appended to that element. Regardless of the value, if no element is found, the menu will be appended to the body.

#### Code examples:

Initialize the autocomplete with the `appendTo` option specified:

```
1 | $( ".selector" ).autocomplete({ appendTo
```

Get or set the `appendTo` option, after initialization:

```
1 | // getter
2 | var appendTo = $( ".selector" ).autocomp
3 |
4 | // setter
5 | $( ".selector" ).autocomplete( "option",
```

### autoFocus

Type: [Boolean](#)

Default: `false`

If set to `true` the first item will automatically be focused when the menu is shown.

### Code examples:

Initialize the autocomplete with the autoFocus option specified:

```
1 | $( ".selector" ).autocomplete({ autoFocus: true
```

Get or set the autoFocus option, after initialization:

```
1 | // getter
2 | var autoFocus = $( ".selector" ).autocomplete(
3 |
4 | // setter
5 | $( ".selector" ).autocomplete( "option", "autoFocus"
```

## delay

Type: Integer

Default: 300

The delay in milliseconds between when a keystroke occurs and when a search is performed. A zero-delay makes sense for local data (more responsive), but can produce a lot of load for remote data, while being less responsive.

### Code examples:

Initialize the autocomplete with the delay option specified:

```
1 | $( ".selector" ).autocomplete({ delay: 500 });
```

Get or set the delay option, after initialization:

```
1 | // getter
2 | var delay = $( ".selector" ).autocomplete( "option"
```

```
3
4 // setter
5 $( ".selector" ).autocomplete( "option", "delay"
```

## disabled

Type: [Boolean](#)

Default: `false`

Disables the autocomplete if set to `true`.

### Code examples:

Initialize the autocomplete with the disabled option specified:

```
1 | $( ".selector" ).autocomplete({ disabled: true
```

Get or set the disabled option, after initialization:

```
1 // getter
2 var disabled = $( ".selector" ).autocomplete(
3
4 // setter
5 $( ".selector" ).autocomplete( "option", "disabled"
```

## minLength

Type: [Integer](#)

Default: `1`

The minimum number of characters a user must type before a search is performed. Zero is useful for local data with just a few items, but a higher value should be used when a single character search could match a few thousand items.

### Code examples:

Initialize the autocomplete with the minLength option

specified:

```
1 | $( ".selector" ).autocomplete({ minLength: 0 });
```

Get or set the minLength option, after initialization:

```
1 | // getter
2 | var minLength = $( ".selector" ).autocomplete(
3 |
4 | // setter
5 | $( ".selector" ).autocomplete( "option", "minLength"
```

## position

Type: [Object](#)

Default: `{ my: "left top", at: "left bottom", collision: "none" }`

Identifies the position of the suggestions menu in relation to the associated input element. The `of` option defaults to the input element, but you can specify another element to position against. You can refer to the [jQuery UI Position](#) utility for more details about the various options.

### Code examples:

Initialize the autocomplete with the position option specified:

```
1 | $( ".selector" ).autocomplete({ position: { my :
```

Get or set the position option, after initialization:

```
1 | // getter
2 | var position = $( ".selector" ).autocomplete(
```

```
3
4 // setter
5 $( ".selector" ).autocomplete( "option", "position"
```

## source

**Type:** [Array](#) or [String](#) or [Function](#)( [Object](#) request, [Function](#) response( [Object](#) data ) )

**Default:** none; must be specified

Defines the data to use, must be specified.

Independent of the variant you use, the label is always treated as text. If you want the label to be treated as html you can use [Scott González' html extension](#). The demos all focus on different variations of the `source` option - look for one that matches your use case, and check out the code.

### Multiple types supported:

**Array:** An array can be used for local data. There are two supported formats:

An array of strings: [ "Choice1", "Choice2" ]

An array of objects with `label` and `value` properties: [ { label: "Choice1", value: "value1" }, ... ]

The label property is displayed in the suggestion menu. The value will be inserted into the input element when a user selects an item. If just one property is specified, it will be used for both, e.g., if you provide only `value` properties, the value will also be used as the label.

**String:** When a string is used, the Autocomplete plugin expects that string to point to a URL resource that will return JSON data. It can be on the same host or on a different one (must provide JSONP). The Autocomplete plugin does not filter the results, instead a query string is added with a `term` field, which the server-side script should use for filtering the results. For example, if the `source` option is set to "http://example.com" and the user types `foo`, a GET request would be made to

`http://example.com?term=foo`. The data itself can be in the same format as the local data described above.

**Function:** The third variation, a callback, provides the most flexibility and can be used to connect any data source to Autocomplete. The callback gets two arguments:

A `request` object, with a single `term` property, which refers to the value currently in the text input. For example, if the user enters `"new yo"` in a city field, the Autocomplete term will equal `"new yo"`.

A `response` callback, which expects a single argument: the data to suggest to the user. This data should be filtered based on the provided term, and can be in any of the formats described above for simple local data. It's important when providing a custom source callback to handle errors during the request. You must always call the `response` callback even if you encounter an error. This ensures that the widget always has the correct state.

When filtering data locally, you can make use of the built-in `$.ui.autocomplete.escapeRegex` function. It'll take a single string argument and escape all regex characters, making the result safe to pass to `new RegExp()`.

### Code examples:

Initialize the autocomplete with the source option specified:

```
1 | $( ".selector" ).autocomplete({ source: [ "C++" ]
```

Get or set the source option, after initialization:

```
1 | // getter  
2 | var source = $( ".selector" ).autocomplete( "option"  
3 |
```

```
4 // setter  
5 $( ".selector" ).autocomplete( "option", "source"
```

## Methods

### close()

Closes the Autocomplete menu. Useful in combination with the [search](#) method, to close the open menu.

This method does not accept any arguments.

#### Code examples:

Invoke the close method:

```
1 | $( ".selector" ).autocomplete( "close"
```

### destroy()

Removes the autocomplete functionality completely. This will return the element back to its pre-init state.

This method does not accept any arguments.

#### Code examples:

Invoke the destroy method:

```
1 | $( ".selector" ).autocomplete( "destroy"
```

### disable()

Disables the autocomplete.

This method does not accept any arguments.

### Code examples:

Invoke the disable method:

```
1 | $( ".selector" ).autocomplete( "disable" );
```

## enable()

Enables the autocomplete.

This method does not accept any arguments.

### Code examples:

Invoke the enable method:

```
1 | $( ".selector" ).autocomplete( "enable" );
```

## option( optionName )

Returns: [Object](#)

Gets the value currently associated with the specified `optionName`.

### **optionName**

Type: [String](#)

The name of the option to get.

### Code examples:

Invoke the method:

```
1 | var isDisabled = $( ".selector" ).autocomplete(
```

## option()

Returns: [PlainObject](#)

Gets an object containing key/value pairs representing the current autocomplete options hash.

This method does not accept any arguments.

### Code examples:

Invoke the method:

```
1 | var options = $( ".selector" ).autocomplete(
```

## option( optionName, value )

Sets the value of the autocomplete option associated with the specified `optionName`.

### optionName

Type: [String](#)

The name of the option to set.

### value

Type: [Object](#)

A value to set for the option.

### Code examples:

Invoke the method:

```
1 | $( ".selector" ).autocomplete( "option", "disabled"
```

## option( options )

Sets one or more options for the autocomplete.

### options

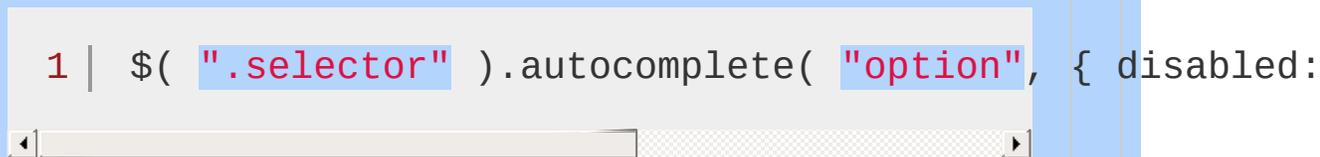
Type: [Object](#)

A map of option-value pairs to set.

### Code examples:

Invoke the method:

```
1 | $( ".selector" ).autocomplete( "option", { disabled:
```



## search( [value ] )

Triggers a [search](#) event and invokes the data source if the event is not canceled. Can be used by a selectbox-like button to open the suggestions when clicked. When invoked with no parameters, the current input's value is used. Can be called with an empty string and `minLength: 0` to display all items.

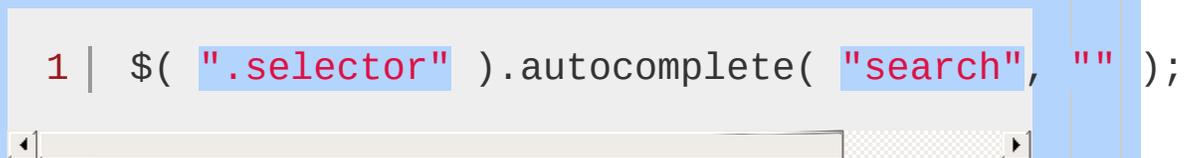
### value

Type: [String](#)

### Code examples:

Invoke the search method:

```
1 | $( ".selector" ).autocomplete( "search", "" );
```



## widget()

Returns: [jQuery](#)

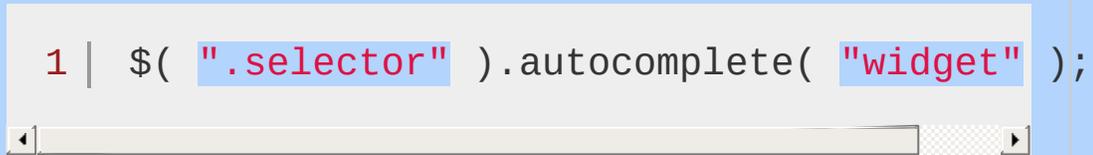
Returns a `jQuery` object containing the menu element. Although the menu items are constantly created and destroyed, the menu element itself is created during initialization and is constantly reused.

This method does not accept any arguments.

**Code examples:**

Invoke the `widget` method:

```
1 | $( ".selector" ).autocomplete( "widget" );
```



## Events

**change( event, ui )**      *Type:* `autocompletechange`

Triggered when the field is blurred, if the value has changed.

**event**

Type: `Event`

**ui**

Type: `Object`

**item**

Type: `jQuery`

The item selected from the menu, if any. Otherwise the property is `null`.

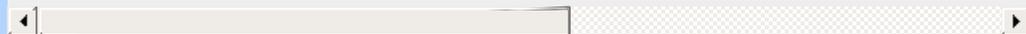
### Code examples:

Initialize the autocomplete with the change callback specified:

```
1 | $( ".selector" ).autocomplete({  
2 |   change: function( event, ui ) {}  
3 | });
```

Bind an event listener to the autocompletechange event:

```
1 | $( ".selector" ).on( "autocompletechange
```



**close( event, ui )**      *Type:* `autocompleteclose`

Triggered when the menu is hidden. Not every `close` event will be accompanied by a `change` event.

**event**

Type: [Event](#)

**ui**

Type: [Object](#)

### Code examples:

Initialize the autocomplete with the close callback specified:

```
1 | $( ".selector" ).autocomplete({  
2 |   close: function( event, ui ) {}  
3 | });
```

Bind an event listener to the autocompleteclose event:

```
1 | $( ".selector" ).on( "autocompleteclose", function
```



**create( event, ui )**

Type: `autocompletecreate`

Triggered when the autocomplete is created.

**event**

Type: [Event](#)

**ui**

Type: [Object](#)

### Code examples:

Initialize the autocomplete with the create callback specified:

```
1 | $( ".selector" ).autocomplete({
```

```
2 |     create: function( event, ui ) {}
3 | });
```

Bind an event listener to the autocompletecreate event:

```
1 | $( ".selector" ).on( "autocompletecreate", function
```

## focus( event, ui )

Type: `autocompletefocus`

Triggered when focus is moved to an item (not selecting). The default action is to replace the text field's value with the value of the focused item, though only if the event was triggered by a keyboard interaction. Canceling this event prevents the value from being updated, but does not prevent the menu item from being focused.

### event

Type: [Event](#)

### ui

Type: [Object](#)

#### item

Type: [jQuery](#)

The focused item.

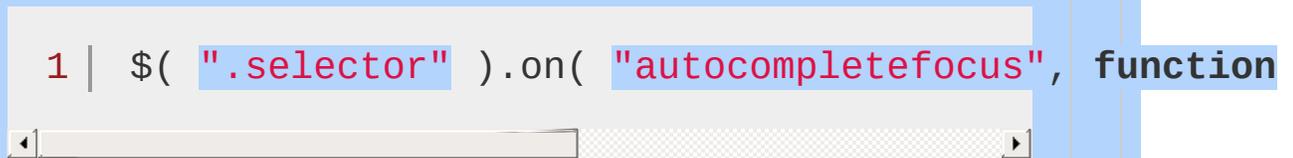
## Code examples:

Initialize the autocomplete with the focus callback specified:

```
1 | $( ".selector" ).autocomplete({
2 |     focus: function( event, ui ) {}
3 | });
```

Bind an event listener to the autocompletefocus event:

```
1 | $( ".selector" ).on( "autocompletefocus", function
```



**open( event, ui )** *Type:* autocompleteopen

Triggered when the suggestion menu is opened or updated.

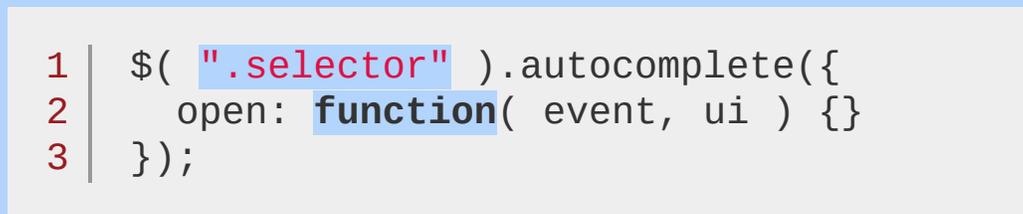
**event**  
*Type:* [Event](#)

**ui**  
*Type:* [Object](#)

### Code examples:

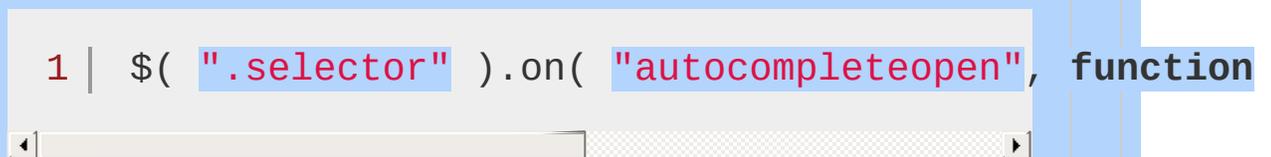
Initialize the autocomplete with the open callback specified:

```
1 | $( ".selector" ).autocomplete({  
2 |   open: function( event, ui ) {}  
3 | });
```



Bind an event listener to the autocompleteopen event:

```
1 | $( ".selector" ).on( "autocompleteopen", function
```



**response( event, ui )** *Type:* autocompleteresponse

Triggered after a search completes, before the menu is shown. Useful for local manipulation of suggestion data, where a custom [source](#) option callback is not required. This

event is always triggered when a search completes, even if the menu will not be shown because there are no results or the Autocomplete is disabled.

**event**

Type: `Event`

**ui**

Type: `Object`

**content**

Type: `Array`

Contains the response data and can be modified to change the results that will be shown. This data is already normalized, so if you modify the data, make sure to include both `value` and `label` properties for each item.

**Code examples:**

Initialize the autocomplete with the response callback specified:

```
1 | $( ".selector" ).autocomplete({  
2 |   response: function( event, ui ) {}  
3 | });
```

Bind an event listener to the autocomplete response event:

```
1 | $( ".selector" ).on( "autocomplete response",
```



**search( event, ui )**

Type: `autocomplete search`

Triggered before a search is performed, after `minLength` and `delay` are met. If canceled, then no request will be started and no items suggested.

**event**  
Type: [Event](#)

**ui**  
Type: [Object](#)

### Code examples:

Initialize the autocomplete with the search callback specified:

```
1 | $( ".selector" ).autocomplete({  
2 |   search: function( event, ui ) {}  
3 | });
```

Bind an event listener to the autocomplete search event:

```
1 | $( ".selector" ).on( "autocomplete search", function
```



---

**select( event, ui )**      Type: [autocomplete select](#)

Triggered when an item is selected from the menu. The default action is to replace the text field's value with the value of the selected item.

Canceling this event prevents the value from being updated, but does not prevent the menu from closing.

**event**  
Type: [Event](#)

**ui**  
Type: [Object](#)

**item**  
Type: [jQuery](#)  
The selected item.

### Code examples:

Initialize the autocomplete with the select callback specified:

```
1 | $( ".selector" ).autocomplete({  
2 |   select: function( event, ui ) {}  
3 | });
```

Bind an event listener to the autocomplete select event:

```
1 | $( ".selector" ).on( "autocompleteselect", function
```



## Examples:

### Example: *A simple jQuery UI Autocomplete*

```
1 | <!doctype html>
2 | <html lang="en">
3 | <head>
4 |   <meta charset="utf-8">
5 |   <title>autocomplete demo</title>
6 |   <link rel="stylesheet" href="http://code.jq
7 |   <script src="http://code.jquery.com/jquery-:
8 |   <script src="http://code.jquery.com/ui/1.10
9 | </head>
10 | <body>
11 |
12 | <label for="autocomplete">Select a programmin
13 | <input id="autocomplete">
14 |
15 | <script>
16 | $( "#autocomplete" ).autocomplete({
17 |   source: [ "c++", "java", "php", "coldfusion
18 | });
19 | </script>
20 |
21 | </body>
22 | </html>
```

### Demo

**Example:** *Using a custom source callback to match only the beginning of terms*

```
1 | <!doctype html>
2 | <html lang="en">
```

```
3 <head>
4   <meta charset="utf-8">
5   <title>autocomplete demo</title>
6   <link rel="stylesheet" href="http://code.jquery.com/u
7   <script src="http://code.jquery.com/jquery-1.9.1.js"
8   <script src="http://code.jquery.com/ui/1.10.1/jquery-
9 </head>
10 <body>
11
12 <label for="autocomplete">Select a programming language
13 <input id="autocomplete">
14
15 <script>
16 var tags = [ "c++", "java", "php", "coldfusion"
17 $( "#autocomplete" ).autocomplete({
18   source: function( request, response ) {
19     var matcher = new RegExp( "^" + $.ui.autocomp
20     response( $.grep( tags, function( item ) {
21       return matcher.test( item );
22     } ) );
23   }
24 });
25 </script>
26
27 </body>
28 </html>
```

Demo

**A new version of this book is available!**

---

# Blind Effect

Categories: [Effects](#)

## Blind Effect

**Description:** *The blind effect hides or shows an element by wrapping the element in a container, and "pulling the blinds"*

### blind

**direction** (default: "up")

Type: String

The direction the blind will be pulled to hide the element, or the direction from which the element will be revealed.

Possible Values: up, down, left, right, vertical, horizontal.

The container has `overflow: hidden` applied, so height changes affect what's visible.

## Example:

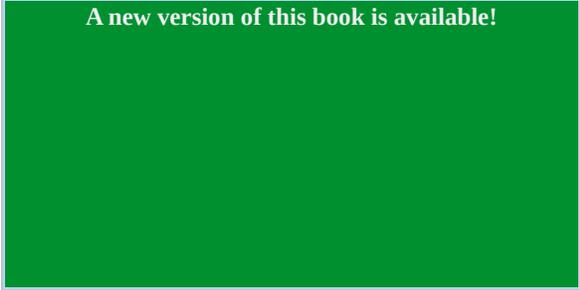
*Toggle a div using the blind effect.*

```
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>blind demo</title>
6   <link rel="stylesheet" href="http://code.jq
7   <style>
8     #toggle {
9       width: 100px;
10      height: 100px;
11      background: #ccc;
12    }
13  </style>
14  <script src="http://code.jquery.com/jquery-:
15  <script src="http://code.jquery.com/ui/1.10
16 </head>
17 <body>
18
19 <p>Click anywhere to toggle the box.</p>
20 <div id="toggle"></div>
21
22 <script>
23 $( document ).click(function() {
24   $( "#toggle" ).toggle( "blind" );
25 });
26 </script>
27
28 </body>
29 </html>
```

Demo

POWERED BYHERONOTE

A new version of this book is available!



---

# Bounce Effect

Categories: [Effects](#)

## Bounce Effect

**Description:** *The bounce effect bounces an element. When used with hide or show, the last or first bounce will also fade in/out.*

### bounce

**distance** (default: 20)

Type: Number

The distance of the largest "bounce" in pixels.

**times** (default: 5)

Type: Integer

The number of times the element will bounce. When used with hide or show, there is an extra "half" bounce for the fade in/out.

## Example:

*Toggle a div using the bounce effect.*

```
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>bounce demo</title>
6   <link rel="stylesheet" href="http://code.jq
7   <style>
8     #toggle {
9       width: 100px;
10      height: 100px;
11      background: #ccc;
12    }
13  </style>
14  <script src="http://code.jquery.com/jquery-:
15  <script src="http://code.jquery.com/ui/1.10
16 </head>
17 <body>
18
19 <p>Click anywhere to toggle the box.</p>
20 <div id="toggle"></div>
21
22 <script>
23 $( document ).click(function() {
24   $( "#toggle" ).toggle( "bounce", { times: 3
25 });
26 </script>
27
28 </body>
29 </html>
```

Demo

POWERED BYHERONOTE

A new version of this book is available!

---

# Button Widget

Categories: [Widgets](#)

---

Button Widget

version added: 1.8

**Description:** *Themable buttons and button sets.*

## QuickNav

## Examples

### Options

[disabled](#)  
[icons](#)  
[label](#)  
[text](#)

### Methods

[destroy](#)  
[disable](#)  
[enable](#)  
[refresh](#)  
[option](#)  
[widget](#)

### Events

[create](#)

Button enhances standard form elements like buttons, inputs and anchors to themable buttons with appropriate hover and active styles.

In addition to basic push buttons, radio buttons and checkboxes (inputs of type radio and checkbox) can be converted to buttons. Their associated label is styled to appear as the button, while the underlying input is updated on click. For the association to work properly, give the input an `id` attribute, and refer to that in the label's `for` attribute. Don't nest the input inside the label, as that [causes accessibility problems](#).

In order to group radio buttons, Button also provides an additional widget, called Buttonset. Buttonset is used by selecting a container element (which contains the radio buttons) and calling `.buttonset()`. Buttonset will also provide visual grouping, and therefore should be used whenever you have a group of buttons. It works by selecting all descendants and applying `.button()` to them. You can enable and disable a button set, which will enable and disable all contained buttons. Destroying a button set also calls each button's `destroy` method.

When using an input of type button, submit or reset, support is limited to plain text labels with no icons.

## Dependencies

[UI Core](#)

## Widget Factory

### **Additional Notes:**

This widget requires some functional CSS, otherwise it won't work. If you build a custom theme, use the widget's specific CSS file as a starting point.

## Options

### disabled

Type: [Boolean](#)

Default: `false`

Disables the button if set to `true`.

#### Code examples:

Initialize the button with the disabled option specified:

```
1 | $( ".selector" ).button({ disabled: true
```

Get or set the disabled option, after initialization:

```
1 | // getter
2 | var disabled = $( ".selector" ).button(
3 |
4 | // setter
5 | $( ".selector" ).button( "option", "dis
```

### icons

Type: [Object](#)

Default: `{ primary: null, secondary: null }`

Icons to display, with or without text (see [text](#) option). By default, the primary icon is displayed on the left of the label text and the secondary is displayed on the right. The positioning can be controlled via CSS. The value for the `primary` and `secondary` properties must be a class name, e.g., `"ui-icon-gear"`. For using only one icon: `icons: { primary: "ui-icon-locked" }`. For using two icons: `icons: { primary: "ui-icon-gear", secondary: "ui-icon-triangle-1-s"`

```
}.
```

### Code examples:

Initialize the button with the icons option specified:

```
1 | $( ".selector" ).button({ icons: { primary: "ui-icon-
```

Get or set the icons option, after initialization:

```
1 | // getter
2 | var icons = $( ".selector" ).button( "option"
3 |
4 | // setter
5 | $( ".selector" ).button( "option", "icons", { primary
```

## label

Type: [String](#)

Default: `null`

Text to show in the button. When not specified (`null`), the element's HTML content is used, or its `value` attribute if the element is an input element of type submit or reset, or the HTML content of the associated label element if the element is an input of type radio or checkbox.

### Code examples:

Initialize the button with the label option specified:

```
1 | $( ".selector" ).button({ label: "custom label"
```

Get or set the label option, after initialization:

```
1 | // getter
```

```
2 var label = $( ".selector" ).button( "option"  
3  
4 // setter  
5 $( ".selector" ).button( "option", "label", "custom I
```

## text

Type: [Boolean](#)

Default: [true](#)

Whether to show the label. When set to [false](#) no text will be displayed, but the [icons](#) option must be enabled, otherwise the [text](#) option will be ignored.

### Code examples:

Initialize the button with the text option specified:

```
1 | $( ".selector" ).button({ text: false });
```

Get or set the text option, after initialization:

```
1 | // getter  
2 | var text = $( ".selector" ).button( "option",  
3 |  
4 | // setter  
5 | $( ".selector" ).button( "option", "text", false
```

## Methods

### destroy()

Removes the button functionality completely. This will return the element back to its pre-init state.

This method does not accept any arguments.

#### Code examples:

Invoke the destroy method:

```
1 | $( ".selector" ).button( "destroy" );
```

### disable()

Disables the button.

This method does not accept any arguments.

#### Code examples:

Invoke the disable method:

```
1 | $( ".selector" ).button( "disable" );
```

### enable()

Enables the button.

This method does not accept any arguments.

### Code examples:

Invoke the enable method:

```
1 | $( ".selector" ).button( "enable" );
```

## option( optionName )

Returns: *Object*

Gets the value currently associated with the specified `optionName`.

**optionName**

Type: *String*

The name of the option to get.

### Code examples:

Invoke the method:

```
1 | var isDisabled = $( ".selector" ).button( "option"
```

## option()

Returns: *PlainObject*

Gets an object containing key/value pairs representing the current button options hash.

This method does not accept any arguments.

### Code examples:

Invoke the method:

```
1 | var options = $( ".selector" ).button( "option"
```

## option( optionName, value )

Sets the value of the button option associated with the specified `optionName`.

### **optionName**

Type: String

The name of the option to set.

### **value**

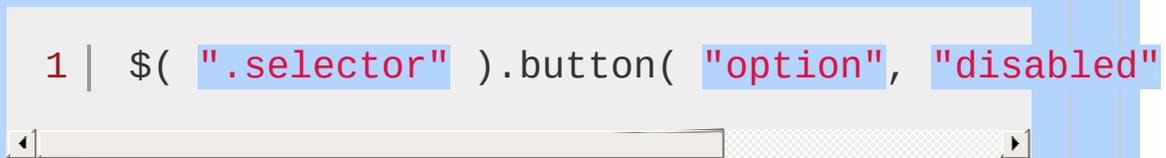
Type: Object

A value to set for the option.

### **Code examples:**

Invoke the method:

```
1 | $( ".selector" ).button( "option", "disabled"
```



## option( options )

Sets one or more options for the button.

### **options**

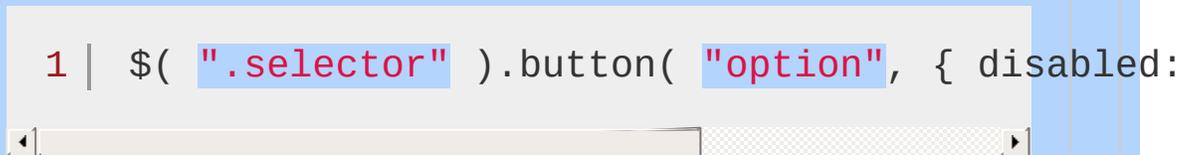
Type: Object

A map of option-value pairs to set.

### **Code examples:**

Invoke the method:

```
1 | $( ".selector" ).button( "option", { disabled:
```



## refresh()

Refreshes the visual state of the button. Useful for updating button state after the native element's checked or disabled state is changed programmatically.

This method does not accept any arguments.

### Code examples:

Invoke the refresh method:

```
1 | $( ".selector" ).button( "refresh" );
```

## widget()

Returns: *jQuery*

Returns a `jQuery` object containing the element visually representing the button.

This method does not accept any arguments.

### Code examples:

Invoke the widget method:

```
1 | var widget = $( ".selector" ).button( "widget" );
```



## Events

**create( event, ui )**

Type: `buttoncreate`

Triggered when the button is created.

**event**

Type: `Event`

**ui**

Type: `Object`

### Code examples:

Initialize the button with the create callback specified:

```
1 | $( ".selector" ).button({  
2 |   create: function( event, ui ) {}  
3 | });
```

Bind an event listener to the buttoncreate event:

```
1 | $( ".selector" ).on( "buttoncreate", fu
```

## Examples:

### Example: *A simple jQuery UI Button*

```
1  <!doctype html>
2  <html lang="en">
3  <head>
4      <meta charset="utf-8">
5      <title>button demo</title>
6      <link rel="stylesheet" href="http://code.jq
7      <script src="http://code.jquery.com/jquery-:
8      <script src="http://code.jquery.com/ui/1.10
9  </head>
10 <body>
11
12 <button>Button label</button>
13
14 <script>
15 $( "button" ).button();
16 </script>
17
18 </body>
19 </html>
```

Demo

### Example: *A simple jQuery UI Buttonset*

```
1  <!doctype html>
2  <html lang="en">
3  <head>
4      <meta charset="utf-8">
5      <title>button demo</title>
6      <link rel="stylesheet" href="http://code.jq
```

```
7 <script src="http://code.jquery.com/jquery-1.9.1.js"
8 <script src="http://code.jquery.com/ui/1.10.1/jquery-
9 </head>
10 <body>
11
12 <div id="radio">
13 <input type="radio" id="radio1" name="radio">
14 <input type="radio" id="radio2" name="radio"
15 <input type="radio" id="radio3" name="radio">
16 </div>
17
18 <script>
19 $( "#radio" ).buttonset();
20 </script>
21
22 </body>
23 </html>
```

Demo

POWERED BYHERONOTE

A new version of this book is available!

---

# Clip Effect

Categories: [Effects](#)

## Clip Effect

**Description:** *The clip effect will hide or show an element by clipping the element vertically or horizontally.*

### clip

**direction** (default: "up")

Type: String

The plane in which the clip effect will hide or show its element.

`vertical` clips the top and bottom edges, while `horizontal` clips the right and left edges.

## Example:

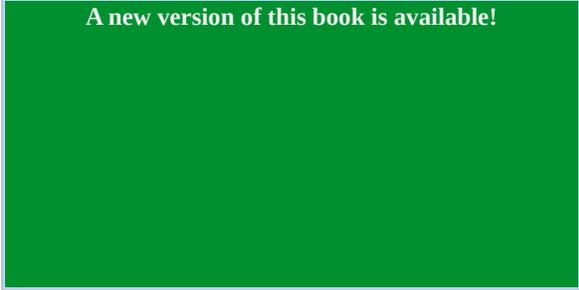
*Toggle a div using the clip effect.*

```
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>clip demo</title>
6   <link rel="stylesheet" href="http://code.jq
7   <style>
8     #toggle {
9       width: 100px;
10      height: 100px;
11      background: #ccc;
12    }
13  </style>
14  <script src="http://code.jquery.com/jquery-:
15  <script src="http://code.jquery.com/ui/1.10
16 </head>
17 <body>
18
19 <p>Click anywhere to toggle the box.</p>
20 <div id="toggle"></div>
21
22 <script>
23 $( document ).click(function() {
24   $( "#toggle" ).toggle( "clip" );
25 });
26 </script>
27
28 </body>
29 </html>
```

Demo

POWERED BYHERONOTE

A new version of this book is available!



---

# :data() Selector

Categories: [Selectors](#) | [UI Core](#)

## data selector

**Description:** *Selects elements which have data stored under the specified key.*

```
jQuery( ":data(key)" )
```

**key:** The data key.

The expression `$( "div:data(foo)" )` matches a `<div>` if it has data stored via `.data( "foo", value )`.

## Example:

*Select elements with data and change their background color.*

```
1  <!doctype html>
2  <html lang="en">
3  <head>
4      <meta charset="utf-8">
5      <title>data demo</title>
6      <link rel="stylesheet" href="http://code.jq
7      <style>
8      div {
9          width: 100px;
10         height: 100px;
11         border: 1px solid #000;
12         float: left;
13     }
14 </style>
15 <script src="http://code.jquery.com/jquery-:
16 <script src="http://code.jquery.com/ui/1.10
17 </head>
18 <body>
19
20 <div id="one"></div>
21 <div id="two"></div>
22 <div id="three"></div>
23
24 <script>
25 $( "#one" ).data( "color", "blue" );
26 $( "#three" ).data( "color", "green" );
27
28 $( ":data(color)" ).each( function() {
29     var element = $( this );
30     element.css( "backgroundColor", element.data
```

```
31  });  
32  </script>  
33  
34  </body>  
35  </html>
```

Demo

POWERED BYHERONOTE

A new version of this book is available!

---

# Datepicker Widget

Categories: [Widgets](#)

---

Datepicker Widget

version added: 1.0

**Description:** *Select a date from a popup or inline calendar*

## QuickNav

## Examples

### Options

[altField](#)  
[altFormat](#)  
[appendText](#)  
[autoSize](#)  
[buttonImage](#)  
[buttonImageOnly](#)  
[buttonText](#)  
[calculateWeek](#)  
[changeMonth](#)  
[changeYear](#)  
[closeText](#)  
[constrainInput](#)  
[currentText](#)  
[dateFormat](#)  
[dayNames](#)  
[dayNamesMin](#)  
[dayNamesShort](#)  
[defaultDate](#)  
[duration](#)  
[firstDay](#)  
[gotoCurrent](#)  
[hideIfNoPrevNext](#)  
[isRTL](#)  
[maxDate](#)  
[minDate](#)  
[monthNames](#)  
[monthNamesShort](#)  
[navigationAsDateFormat](#)  
[nextText](#)  
[numberOfMonths](#)  
[prevText](#)  
[selectOtherMonths](#)  
[shortYearCutoff](#)

### Methods

[destroy](#)  
[dialog](#)  
[isDisabled](#)  
[hide](#)  
[show](#)  
[refresh](#)  
[getDate](#)  
[setDate](#)  
[option](#)  
[widget](#)

### Events

[showAnim](#)  
[showButtonPanel](#)  
[showCurrentAtPos](#)  
[showMonthAfterYear](#)  
[showOn](#)  
[showOptions](#)  
[showOtherMonths](#)  
[showWeek](#)  
[stepMonths](#)  
[weekHeader](#)  
[yearRange](#)  
[yearSuffix](#)  
[beforeShow](#)  
[beforeShowDay](#)  
[onChangeMonthYear](#)  
[onClose](#)  
[onSelect](#)

The jQuery UI Datepicker is a highly configurable plugin that adds datepicker functionality to your pages. You can customize the date format and language, restrict the selectable date ranges and add in buttons and other navigation options easily.

By default, the datepicker calendar opens in a small overlay when the associated text field gains focus. For an inline calendar, simply attach the datepicker to a div or span.

## Keyboard interaction

While the datepicker is open, the following key commands are available:

PAGE UP: Move to the previous month.

PAGE DOWN: Move to the next month.

CTRL+PAGE UP: Move to the previous year.

CTRL+PAGE DOWN: Move to the next year.

CTRL+HOME: Move to the current month. Open the datepicker if closed.

CTRL+LEFT: Move to the previous day.

CTRL+RIGHT: Move to the next day.  
CTRL+UP: Move to the previous week.  
CTRL+DOWN: Move the next week.  
ENTER: Select the focused date.  
CTRL+END: Close the datepicker and erase the date.  
ESCAPE: Close the datepicker without selection.

## Utility functions

### **\$.datepicker.setDefaults( settings )**

Change the default settings for all date pickers.

Use the `option()` method to change settings for individual instances.

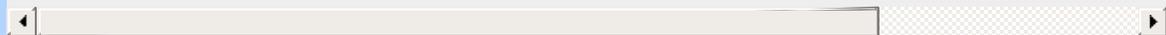
#### **Code examples:**

Set all date pickers to open on focus or a click on an icon.

```
1 | $.datepicker.setDefaults({
2 |   showOn: "both",
3 |   buttonImageOnly: true,
4 |   buttonImage: "calendar.gif",
5 |   buttonText: "Calendar"
6 | });
```

Set all date pickers to have French text.

```
1 | $.datepicker.setDefaults( $.datepicker.regional[
```



### **\$.datepicker.formatDate( format, date, settings )**

Format a date into a string value with a specified format.

The format can be combinations of the following:

d - day of month (no leading zero)  
dd - day of month (two digit)  
o - day of the year (no leading zeros)  
oo - day of the year (three digit)  
D - day name short  
DD - day name long  
m - month of year (no leading zero)  
mm - month of year (two digit)  
M - month name short  
MM - month name long  
y - year (two digit)  
yy - year (four digit)  
@ - Unix timestamp (ms since 01/01/1970)  
! - Windows ticks (100ns since 01/01/0001)  
'...' - literal text  
" - single quote  
anything else - literal text

There are also a number of predefined standard date formats available from `$.datepicker`:

ATOM - 'yy-mm-dd' (Same as RFC 3339/ISO 8601)  
COOKIE - 'D, dd M yy'  
ISO\_8601 - 'yy-mm-dd'  
RFC\_822 - 'D, d M y' (See RFC 822)  
RFC\_850 - 'DD, dd-M-y' (See RFC 850)  
RFC\_1036 - 'D, d M y' (See RFC 1036)  
RFC\_1123 - 'D, d M yy' (See RFC 1123)  
RFC\_2822 - 'D, d M yy' (See RFC 2822)  
RSS - 'D, d M y' (Same as RFC 822)  
TICKS - '!'

TIMESTAMP - '@'

W3C - 'yy-mm-dd' (Same as ISO 8601)

### Code examples:

Display the date in ISO format. Produces "2007-01-26".

```
1 | $.datepicker.formatDate( "yy-mm-dd", new Date
```

Display the date in expanded French format. Produces "Samedi, Juillet 14, 2007".

```
1 | $.datepicker.formatDate( "DD, MM d, yy", new Date  
2 |     dayNamesShort: $.datepicker.regional[ "fr" ].dayNamesShort,  
3 |     dayNames: $.datepicker.regional[ "fr" ].dayNames,  
4 |     monthNamesShort: $.datepicker.regional[ "fr" ].monthNamesShort,  
5 |     monthNames: $.datepicker.regional[ "fr" ].monthNames,  
6 | });
```

### **\$.datepicker.parseDate( format, value, settings )**

Extract a date from a string value with a specified format.

The format can be combinations of the following:

d - day of month (no leading zero)

dd - day of month (two digit)

o - day of year (no leading zeros)

oo - day of year (three digit)

D - day name short

DD - day name long

m - month of year (no leading zero)

mm - month of year (two digit)

M - month name short  
MM - month name long  
y - year (two digit)  
yy - year (four digit)  
@ - Unix timestamp (ms since 01/01/1970)  
! - Windows ticks (100ns since 01/01/0001)  
'...' - literal text  
" - single quote  
anything else - literal text

A number of exceptions may be thrown:

'Invalid arguments' if either format or value is null  
'Missing number at position nn' if format indicated a numeric value that is not then found  
'Unknown name at position nn' if format indicated day or month name that is not then found  
'Unexpected literal at position nn' if format indicated a literal value that is not then found  
'Invalid date' if the date is invalid, such as '31/02/2007'

### Code examples:

Extract a date in ISO format.

```
1 | $.datepicker.parseDate( "yy-mm-dd", "2007-01-31")
```

Extract a date in expanded French format.

```
1 | $.datepicker.parseDate( "DD, MM d, yy", "Samec",  
2 |   shortYearCuroff: 20,  
3 |   dayNamesShort: $.datepicker.regional[ "fr" ],  
4 |   dayNames: $.datepicker.regional[ "fr" ].dayNames
```

```
5 |     monthNamesShort: $.datepicker.regional[ "fr" ].monthNamesShort
6 |     monthNames: $.datepicker.regional[ "fr" ].monthNames
7 | });
```

## \$.datepicker.iso8601Week( date )

Determine the week of the year for a given date: 1 to 53.

This function uses the ISO 8601 definition of a week: weeks start on a Monday and the first week of the year contains January 4. This means that up to three days from the previous year may be included in the of first week of the current year, and that up to three days from the current year may be included in the last week of the previous year.

This function is the default implementation for the `calculateWeek` option.

### Code examples:

Find the week of the year for a date.

```
1 | $.datepicker.iso8601Week( new Date( 2007, 1 -
```

## \$.datepicker.noWeekends

Set as `beforeShowDay` function to prevent selection of weekends.

We can provide the `noWeekends()` function into the `beforeShowDay` option which will calculate all the weekdays and provide an array of `true/false` values indicating whether a date is selectable.

### Code examples:

Set the DatePicker so no weekend is selectable

```
1 | $( "#datepicker" ).datepicker({
2 |     beforeShowDay: $.datepicker.noWeekends
3 | });
```

## Localization

Datepicker provides support for localizing its content to cater for different languages and date formats. Each localization is contained within its own file with the language code appended to the name, e.g., `jquery.ui.datepicker-fr.js` for French. The desired localization file should be included after the main datepicker code. Each localization file adds its settings to the set of available localizations and automatically applies them as defaults for all instances.

The `$.datepicker.regional` attribute holds an array of localizations, indexed by language code, with `""` referring to the default (English). Each entry is an object with the following attributes: `closeText`, `prevText`, `nextText`, `currentText`, `monthNames`, `monthNamesShort`, `dayNames`, `dayNamesShort`, `dayNamesMin`, `weekHeader`, `dateFormat`, `firstDay`, `isRTL`, `showMonthAfterYear`, and `yearSuffix`.

You can restore the default localizations with:

```
$.datepicker.setDefaults( $.datepicker.regional[ "" ] );
```

And can then override an individual datepicker for a specific locale:

```
$( selector ).datepicker( $.datepicker.regional[ "fr" ] );
```

## Dependencies

[UI Core](#)

[Effects Core](#) (optional; for use with the `showAnim` option)

## Additional Notes:

This widget requires some functional CSS, otherwise it won't work. If you build a custom theme, use the widget's specific CSS file as a starting point.

This widget manipulates its element's value programmatically, therefore a native change may not be fired when the element's value changes.

## Options

**altField**      Type: [Selector](#) or [jQuery](#) or [Element](#)

Default: `""`

An input element that is to be updated with the selected date from the datepicker. Use the [altFormat](#) option to change the format of the date within this field. Leave as blank for no alternate field.

### Code examples:

Initialize the datepicker with the altField option specified:

```
1 | $( ".selector" ).datepicker({ altField:
```

Get or set the altField option, after initialization:

```
1 | // getter
2 | var altField = $( ".selector" ).datepicker
3 |
4 | // setter
5 | $( ".selector" ).datepicker( "option", "
```

**altFormat**      Type: [String](#)

Default: `""`

The [dateFormat](#) to be used for the [altField](#) option. This allows one date format to be shown to the user for selection purposes, while a different format is actually sent behind the scenes. For a full list of the possible formats see the [\[\[UI/Datepicker/formatDate|formatDate\]\]](#) function

### Code examples:

Initialize the datepicker with the altFormat option specified:

```
1 | $( ".selector" ).datepicker({ altFormat: "yy-mm-dd"
```

Get or set the altFormat option, after initialization:

```
1 | // getter
2 | var altFormat = $( ".selector" ).datepicker(
3 |
4 | // setter
5 | $( ".selector" ).datepicker( "option", "altFormat"
```

## appendText

Type: [String](#)

Default: ""

The text to display after each date field, e.g., to show the required format.

### Code examples:

Initialize the datepicker with the appendText option specified:

```
1 | $( ".selector" ).datepicker({ appendText: "(yyyy-mm-d"
```

Get or set the appendText option, after initialization:

```
1 | // getter
2 | var appendText = $( ".selector" ).datepicker(
3 |
4 | // setter
```

```
5 $( ".selector" ).datepicker( "option", "appendText"
```

## autoSize

Type: [Boolean](#)

Default: `false`

Set to `true` to automatically resize the input field to accommodate dates in the current `dateFormat`.

### Code examples:

Initialize the datepicker with the autoSize option specified:

```
1 $( ".selector" ).datepicker({ autoSize: true });
```

Get or set the autoSize option, after initialization:

```
1 // getter
2 var autoSize = $( ".selector" ).datepicker( "option"
3
4 // setter
5 $( ".selector" ).datepicker( "option", "autoSize"
```

## beforeShow

Default: **Type:** [Function](#)( [Element](#) input, [Object](#) inst )  
`null`

A function that takes an input field and current datepicker instance and returns an options object to update the datepicker with. It is called just before the datepicker is displayed.

**beforeShowDay**      **Type:** [Function](#)( [Date](#) date )

**Default:** `null`

A function takes a date as a parameter and must return an array with `[0]` equal to `true/false` indicating whether or not this date is selectable, `[1]` equal to a CSS class name or `""` for the default presentation, and `[2]` an optional popup tooltip for this date. It is called for each day in the datepicker before it is displayed.

**buttonImage**      **Type:** [String](#)

**Default:** `""`

The URL for the popup button image. If set, the `buttonText` option becomes the `alt` value and is not directly displayed.

**Code examples:**

Initialize the datepicker with the `buttonImage` option specified:

```
1 | $( ".selector" ).datepicker({ buttonImage: "/images/0
```

Get or set the `buttonImage` option, after initialization:

```
1 | // getter
2 | var buttonImage = $( ".selector" ).datepicker(
3 |
4 | // setter
5 | $( ".selector" ).datepicker( "option", "buttonImage"
```

**buttonImageOnly**      **Type:** [Boolean](#)

**Default:** `false`

Whether the button image should be rendered by itself instead of inside a button element.

### Code examples:

Initialize the datepicker with the `buttonImageOnly` option specified:

```
1 | $( ".selector" ).datepicker({ buttonImageOnly:
```

Get or set the `buttonImageOnly` option, after initialization:

```
1 | // getter
2 | var buttonImageOnly = $( ".selector" ).datepicker(
3 |
4 | // setter
5 | $( ".selector" ).datepicker( "option", "buttonImageOn
```

## buttonText

Type: String

Default: `"..."`

The text to display on the trigger button. Use in conjunction with the `showOn` option set to `"button"` or `"both"`.

### Code examples:

Initialize the datepicker with the `buttonText` option specified:

```
1 | $( ".selector" ).datepicker({ buttonText: "Choose"
```

Get or set the `buttonText` option, after initialization:

```
1 | // getter
2 | var buttonText = $( ".selector" ).datepicker(
```

```
3
4 // setter
5 $( ".selector" ).datepicker( "option", "buttonText"
```

## calculateWeek

Type: [Function\(\)](#)

Default: `jQuery.datepicker.iso8601Week`

A function to calculate the week of the year for a given date. The default implementation uses the ISO 8601 definition: weeks start on a Monday; the first week of the year contains the first Thursday of the year.

### Code examples:

Initialize the datepicker with the calculateWeek option specified:

```
1 | $( ".selector" ).datepicker({ calculateWeek: myWeekCa
```

Get or set the calculateWeek option, after initialization:

```
1 | // getter
2 | var calculateWeek = $( ".selector" ).datepicker(
3 |
4 | // setter
5 | $( ".selector" ).datepicker( "option", "calculateWeek"
```

## changeMonth

Type: [Boolean](#)

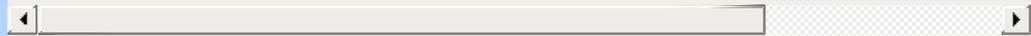
Default: `false`

Whether the month should be rendered as a dropdown instead of text.

### Code examples:

Initialize the datepicker with the `changeMonth` option specified:

```
1 | $( ".selector" ).datepicker({ changeMonth: true
```



Get or set the `changeMonth` option, after initialization:

```
1 | // getter
2 | var changeMonth = $( ".selector" ).datepicker(
3 |
4 | // setter
5 | $( ".selector" ).datepicker( "option", "changeMonth"
```



## changeYear

Type: [Boolean](#)

Default: `false`

Whether the year should be rendered as a dropdown instead of text. Use the [yearRange](#) option to control which years are made available for selection.

### Code examples:

Initialize the datepicker with the `changeYear` option specified:

```
1 | $( ".selector" ).datepicker({ changeYear: true
```



Get or set the `changeYear` option, after initialization:

```
1 | // getter
2 | var changeYear = $( ".selector" ).datepicker(
```

```
3
4 // setter
5 $( ".selector" ).datepicker( "option", "changeYear"
```

## closeText

Type: [String](#)

Default: `"Done"`

The text to display for the close link. Use the [showButtonPanel](#) option to display this button.

### Code examples:

Initialize the datepicker with the closeText option specified:

```
1 | $( ".selector" ).datepicker({ closeText: "Close"
```

Get or set the closeText option, after initialization:

```
1 // getter
2 var closeText = $( ".selector" ).datepicker(
3
4 // setter
5 $( ".selector" ).datepicker( "option", "closeText"
```

## constrainInput

Type: [Boolean](#)

Default: `true`

When `true`, entry in the input field is constrained to those characters allowed by the current [dateFormat](#) option.

### Code examples:

Initialize the datepicker with the constrainInput option specified:

```
1 | $( ".selector" ).datepicker({ constrainInput:
```

Get or set the constrainInput option, after initialization:

```
1 | // getter
2 | var constrainInput = $( ".selector" ).datepicker(
3 |
4 | // setter
5 | $( ".selector" ).datepicker( "option", "constrainInput"
```

## currentText

Type: [String](#)

Default: "Today"

The text to display for the current day link. Use the [showButtonPanel](#) option to display this button.

### Code examples:

Initialize the datepicker with the currentText option specified:

```
1 | $( ".selector" ).datepicker({ currentText: "Now"
```

Get or set the currentText option, after initialization:

```
1 | // getter
2 | var currentText = $( ".selector" ).datepicker(
3 |
4 | // setter
5 | $( ".selector" ).datepicker( "option", "currentText"
```

## dateFormat

Type: [String](#)

Default: `"mm/dd/yy"`

The format for parsed and displayed dates. For a full list of the possible formats see the

[\[\[UI/Datepicker/formatDate|formatDate\]\]](#) function.

### Code examples:

Initialize the datepicker with the dateFormat option specified:

```
1 | $( ".selector" ).datepicker({ dateFormat: "yy-mm-dd"
```

Get or set the dateFormat option, after initialization:

```
1 | // getter
2 | var dateFormat = $( ".selector" ).datepicker(
3 |
4 | // setter
5 | $( ".selector" ).datepicker( "option", "dateFormat"
```

## dayNames

Type: [Array](#)

Default: `[ "Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday" ]`

The list of long day names, starting from Sunday, for use as requested via the [dateFormat](#) option.

### Code examples:

Initialize the datepicker with the dayNames option specified:

```
1 | $( ".selector" ).datepicker({ dayNames: [ "Dimanche"
```

Get or set the dayNames option, after initialization:

```
1 // getter
2 var dayNames = $( ".selector" ).datepicker( "option"
3
4 // setter
5 $( ".selector" ).datepicker( "option", "dayNames"
```

## dayNamesMin

Type: [Array](#)

Default: [ "Su", "Mo", "Tu", "We", "Th", "Fr", "Sa" ]

The list of minimised day names, starting from Sunday, for use as column headers within the datepicker.

### Code examples:

Initialize the datepicker with the dayNamesMin option specified:

```
1 $( ".selector" ).datepicker({ dayNamesMin: [
```

Get or set the dayNamesMin option, after initialization:

```
1 // getter
2 var dayNamesMin = $( ".selector" ).datepicker(
3
4 // setter
5 $( ".selector" ).datepicker( "option", "dayNamesMin"
```

## dayNamesShort

Type: [Array](#)

**Default:** [ "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat" ]

The list of abbreviated day names, starting from Sunday, for use as requested via the `dateFormat` option.

### Code examples:

Initialize the datepicker with the `dayNamesShort` option specified:

```
1 | $( ".selector" ).datepicker({ dayNamesShort: [
```

Get or set the `dayNamesShort` option, after initialization:

```
1 | // getter
2 | var dayNamesShort = $( ".selector" ).datepicker(
3 |
4 | // setter
5 | $( ".selector" ).datepicker( "option", "dayNamesShort"
```

**defaultDate**      **Type:** [Date](#) or [Number](#) or [String](#)

**Default:** `null`

Set the date to highlight on first opening if the field is blank. Specify either an actual date via a `Date` object or as a string in the current `[[UI/Datepicker#option-dateFormat|dateFormat]]`, or a number of days from today (e.g. `+7`) or a string of values and periods ('y' for years, 'm' for months, 'w' for weeks, 'd' for days, e.g. `'+1m +7d'`), or `null` for today.

### Multiple types supported:

**Date:** A date object containing the default date.

**Number:** A number of days from today. For example `2` represents two days from today and `-1` represents yesterday.

**String:** A string in the format defined by the `dateFormat`

option, or a relative date. Relative dates must contain value and period pairs; valid periods are "y" for years, "m" for months, "w" for weeks, and "d" for days. For example, "+1m +7d" represents one month and seven days from today.

### Code examples:

Initialize the datepicker with the defaultDate option specified:

```
1 | $( ".selector" ).datepicker({ defaultDate: +7
```

Get or set the defaultDate option, after initialization:

```
1 | // getter
2 | var defaultDate = $( ".selector" ).datepicker(
3 |
4 | // setter
5 | $( ".selector" ).datepicker( "option", "defaultDate"
```

## duration

Type: or [String](#)

Default: "normal"

Control the speed at which the datepicker appears, it may be a time in milliseconds or a string representing one of the three predefined speeds ("slow", "normal", "fast").

### Code examples:

Initialize the datepicker with the duration option specified:

```
1 | $( ".selector" ).datepicker({ duration: "slow"
```

Get or set the duration option, after initialization:

```
1 | // getter
2 | var duration = $( ".selector" ).datepicker( "option"
3 |
4 | // setter
5 | $( ".selector" ).datepicker( "option", "duration"
```

## firstDay

Type: Integer

Default: `0`

Set the first day of the week: Sunday is `0`, Monday is `1`, etc.

### Code examples:

Initialize the datepicker with the firstDay option specified:

```
1 | $( ".selector" ).datepicker({ firstDay: 1 });
```

Get or set the firstDay option, after initialization:

```
1 | // getter
2 | var firstDay = $( ".selector" ).datepicker( "option"
3 |
4 | // setter
5 | $( ".selector" ).datepicker( "option", "firstDay"
```

## gotoCurrent

Type: Boolean

Default: `false`

When `true`, the current day link moves to the currently selected date instead of today.

### Code examples:

Initialize the datepicker with the gotoCurrent option specified:

```
1 | $( ".selector" ).datepicker({ gotoCurrent: true
```

Get or set the gotoCurrent option, after initialization:

```
1 | // getter
2 | var gotoCurrent = $( ".selector" ).datepicker(
3 |
4 | // setter
5 | $( ".selector" ).datepicker( "option", "gotoCurrent"
```

## hideIfNoPrevNext

Type: [Boolean](#)

Default: `false`

Normally the previous and next links are disabled when not applicable (see the [minDate](#) and [maxDate](#) options). You can hide them altogether by setting this attribute to `true`.

### Code examples:

Initialize the datepicker with the hideIfNoPrevNext option specified:

```
1 | $( ".selector" ).datepicker({ hideIfNoPrevNext:
```

Get or set the hideIfNoPrevNext option, after initialization:

```
1 | // getter
2 | var hideIfNoPrevNext = $( ".selector" ).datepicker(
3 |
```

```
4 // setter
5 $( ".selector" ).datepicker( "option", "hideIfNoPrevM
```

## isRTL

Type: [Boolean](#)

Default: `false`

Whether the current language is drawn from right to left.

### Code examples:

Initialize the datepicker with the isRTL option specified:

```
1 | $( ".selector" ).datepicker({ isRTL: true });
```

Get or set the isRTL option, after initialization:

```
1 // getter
2 var isRTL = $( ".selector" ).datepicker( "option"
3
4 // setter
5 $( ".selector" ).datepicker( "option", "isRTL"
```

## maxDate

Type: [Date](#) or [Number](#) or [String](#)

Default: `null`

The maximum selectable date. When set to `null`, there is no maximum.

### Multiple types supported:

**Date:** A date object containing the maximum date.

**Number:** A number of days from today. For example `2` represents two days from today and `-1` represents

yesterday.

**String:** A string in the format defined by the `dateFormat` option, or a relative date. Relative dates must contain value and period pairs; valid periods are "y" for years, "m" for months, "w" for weeks, and "d" for days. For example, "+1m +7d" represents one month and seven days from today.

### Code examples:

Initialize the datepicker with the `maxDate` option specified:

```
1 | $( ".selector" ).datepicker({ maxDate: "+1m +1w"
◀ |-----▶
```

Get or set the `maxDate` option, after initialization:

```
1 | // getter
2 | var maxDate = $( ".selector" ).datepicker( "option"
3 |
4 | // setter
5 | $( ".selector" ).datepicker( "option", "maxDate"
◀ |-----▶
```

**minDate**                      **Type:** `Date` or `Number` or `String`

**Default:** `null`

The minimum selectable date. When set to `null`, there is no minimum.

### Multiple types supported:

**Date:** A date object containing the minimum date.

**Number:** A number of days from today. For example `2` represents two days from today and `-1` represents yesterday.

**String:** A string in the format defined by the `dateFormat`

option, or a relative date. Relative dates must contain value and period pairs; valid periods are "y" for years, "m" for months, "w" for weeks, and "d" for days. For example, "+1m +7d" represents one month and seven days from today.

### Code examples:

Initialize the datepicker with the minDate option specified:

```
1 | $( ".selector" ).datepicker({ minDate: new Date(
```

Get or set the minDate option, after initialization:

```
1 | // getter
2 | var minDate = $( ".selector" ).datepicker( "option"
3 |
4 | // setter
5 | $( ".selector" ).datepicker( "option", "minDate"
```

## monthNames

Type: [Array](#)

**Default:** [ "January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", "December" ]

The list of full month names, for use as requested via the [dateFormat](#) option.

### Code examples:

Initialize the datepicker with the monthNames option specified:

```
1 | $( ".selector" ).datepicker({ monthNames: [ "Januar"
```

Get or set the monthNames option, after initialization:

```
1 | // getter
2 | var monthNames = $( ".selector" ).datepicker(
3 |
4 | // setter
5 | $( ".selector" ).datepicker( "option", "monthNames"
```

## monthNamesShort

Type: [Array](#)

Default: [ "Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec" ]

The list of abbreviated month names, as used in the month header on each datepicker and as requested via the [dateFormat](#) option.

### Code examples:

Initialize the datepicker with the monthNamesShort option specified:

```
1 | $( ".selector" ).datepicker({ monthNamesShort: [
```

Get or set the monthNamesShort option, after initialization:

```
1 | // getter
2 | var monthNamesShort = $( ".selector" ).datepicker(
3 |
4 | // setter
5 | $( ".selector" ).datepicker( "option", "monthNamesSho
```

## navigationAsDateFormat

Type: [Boolean](#)

Default: `false`

Whether the `prevText` and `nextText` options should be parsed as dates by the `[[UI/Datepicker/formatDate|formatDate]]` function, allowing them to display the target month names for example.

### Code examples:

Initialize the datepicker with the `navigationAsDateFormat` option specified:

```
1 | $( ".selector" ).datepicker({ navigationAsDateFormat
```

Get or set the `navigationAsDateFormat` option, after initialization:

```
1 | // getter
2 | var navigationAsDateFormat = $( ".selector" ).datepic
3 |
4 | // setter
5 | $( ".selector" ).datepicker( "option", "navigationAsD
```

## nextText

Type: [String](#)

Default: `"Next"`

The text to display for the next month link. With the standard ThemeRoller styling, this value is replaced by an icon.

### Code examples:

Initialize the datepicker with the `nextText` option specified:

```
1 | $( ".selector" ).datepicker({ nextText: "Later"
```

Get or set the nextText option, after initialization:

```
1 // getter
2 var nextText = $( ".selector" ).datepicker( "option"
3
4 // setter
5 $( ".selector" ).datepicker( "option", "nextText"
```

**numberOfMonths**      Type: [Number](#) or [Array](#)

Default: `1`

The number of months to show at once.

**Multiple types supported:**

**Number:** The number of months to display in a single row.

**Array:** An array defining the number of rows and columns to display.

**Code examples:**

Initialize the datepicker with the numberOfMonths option specified:

```
1 | $( ".selector" ).datepicker({ numberOfMonths: [
```

Get or set the numberOfMonths option, after initialization:

```
1 // getter
2 var numberOfMonths = $( ".selector" ).datepicker(
3
4 // setter
```

```
5 $( ".selector" ).datepicker( "option", "numberOfMonth
```

## onChangeMonthYear

**Type:** [Function](#)( [Integer](#) year, [Integer](#) month, [Object](#) inst )

**Default:** `null`

Called when the datepicker moves to a new month and/or year. The function receives the selected year, month (1-12), and the datepicker instance as parameters. `this` refers to the associated input field.

## onClose

**Default:** **Type:** [Function](#)( [String](#) dateText, [Object](#) inst ) `null`

Called when the datepicker is closed, whether or not a date is selected. The function receives the selected date as text ("" if none) and the datepicker instance as parameters. `this` refers to the associated input field.

## onSelect

**Default:** **Type:** [Function](#)( [String](#) dateText, [Object](#) inst ) `null`

Called when the datepicker is selected. The function receives the selected date as text and the datepicker instance as parameters. `this` refers to the associated input field.

## prevText

**Type:** [String](#)

**Default:** `"Prev"`

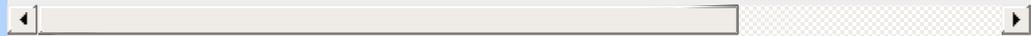
The text to display for the previous month link. With the standard ThemeRoller styling, this value is replaced by an

icon.

### Code examples:

Initialize the datepicker with the `prevText` option specified:

```
1 | $( ".selector" ).datepicker({ prevText: "Earlier"
```



Get or set the `prevText` option, after initialization:

```
1 | // getter
2 | var prevText = $( ".selector" ).datepicker( "option"
3 |
4 | // setter
5 | $( ".selector" ).datepicker( "option", "prevText"
```



## selectOtherMonths

Type: **Boolean**

Default: `false`

Whether days in other months shown before or after the current month are selectable. This only applies if the `showOtherMonths` option is set to `true`.

### Code examples:

Initialize the datepicker with the `selectOtherMonths` option specified:

```
1 | $( ".selector" ).datepicker({ selectOtherMonths:
```



Get or set the `selectOtherMonths` option, after initialization:

```
1 | // getter
2 | var selectOtherMonths = $( ".selector" ).datepicker(
```

```
3
4 // setter
5 $( ".selector" ).datepicker( "option", "selectOtherMo
```

## shortYearCutoff

Type: [Number](#) or [String](#)

Default: `"+10"`

The cutoff year for determining the century for a date (used in conjunction with `[[UI/Datepicker#option-dateFormat|dateFormat]] 'y'`). Any dates entered with a year value less than or equal to the cutoff year are considered to be in the current century, while those greater than it are deemed to be in the previous century.

### Multiple types supported:

**Number:** A value between `0` and `99` indicating the cutoff year.

**String:** A relative number of years from the current year, e.g., `"+3"` or `"-5"`.

### Code examples:

Initialize the datepicker with the shortYearCutoff option specified:

```
1 | $( ".selector" ).datepicker({ shortYearCutoff:
```

Get or set the shortYearCutoff option, after initialization:

```
1 // getter
2 var shortYearCutoff = $( ".selector" ).datepicker(
3
4 // setter
5 $( ".selector" ).datepicker( "option", "shortYearCuto
```

## showAnim

Type: [String](#)

Default: `"show"`

The name of the animation used to show and hide the datepicker. Use `"show"` (the default), `"slideDown"`, `"fadeIn"`, any of the [jQuery UI effects](#). Set to an empty string to disable animation.

### Code examples:

Initialize the datepicker with the showAnim option specified:

```
1 | $( ".selector" ).datepicker({ showAnim: "fold"
```

Get or set the showAnim option, after initialization:

```
1 | // getter
2 | var showAnim = $( ".selector" ).datepicker( "option"
3 |
4 | // setter
5 | $( ".selector" ).datepicker( "option", "showAnim"
```

## showButtonPanel

Type: [Boolean](#)

Default: `false`

Whether to show the button panel.

### Code examples:

Initialize the datepicker with the showButtonPanel option specified:

```
1 | $( ".selector" ).datepicker({ showButtonPanel:
```

Get or set the showButtonPanel option, after initialization:

```
1 // getter
2 var showButtonPanel = $( ".selector" ).datepicker(
3
4 // setter
5 $( ".selector" ).datepicker( "option", "showButtonPanel"
```

## showCurrentAtPos

Type: [Number](#)

Default: `0`

When displaying multiple months via the [numberOfMonths](#) option, the [showCurrentAtPos](#) option defines which position to display the current month in.

### Code examples:

Initialize the datepicker with the showCurrentAtPos option specified:

```
1 $( ".selector" ).datepicker({ showCurrentAtPos:
```

Get or set the showCurrentAtPos option, after initialization:

```
1 // getter
2 var showCurrentAtPos = $( ".selector" ).datepicker(
3
4 // setter
5 $( ".selector" ).datepicker( "option", "showCurrentAt"
```

## showMonthAfterYear

Type: [Boolean](#)

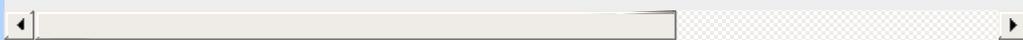
Default: `false`

Whether to show the month after the year in the header.

### Code examples:

Initialize the datepicker with the showMonthAfterYear option specified:

```
1 | $( ".selector" ).datepicker({ showMonthAfterYear:
```



Get or set the showMonthAfterYear option, after initialization:

```
1 | // getter
2 | var showMonthAfterYear = $( ".selector" ).datepicker(
3 |
4 | // setter
5 | $( ".selector" ).datepicker( "option", "showMonthAfterYear",
```



## showOn

Type: [String](#)

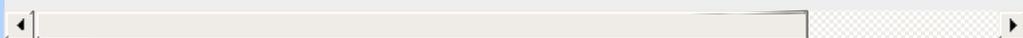
Default: `"focus"`

When the datepicker should appear. The datepicker can appear when the field receives focus (`"focus"`), when a button is clicked (`"button"`), or when either event occurs (`"both"`).

### Code examples:

Initialize the datepicker with the showOn option specified:

```
1 | $( ".selector" ).datepicker({ showOn: "both" });
```



Get or set the showOn option, after initialization:

```
1 | // getter
2 | var showOn = $( ".selector" ).datepicker( "option"
3 |
4 | // setter
5 | $( ".selector" ).datepicker( "option", "showOn"
```

## showOptions

Type: [Object](#)

Default: `{}`

If using one of the jQuery UI effects for the [showAnim](#) option, you can provide additional settings for that animation via this option.

### Code examples:

Initialize the datepicker with the showOptions option specified:

```
1 | $( ".selector" ).datepicker({ showOptions: { directio
```

Get or set the showOptions option, after initialization:

```
1 | // getter
2 | var showOptions = $( ".selector" ).datepicker(
3 |
4 | // setter
5 | $( ".selector" ).datepicker( "option", "showOptions"
```

## showOtherMonths

Type: [Boolean](#)

Default: `false`

Whether to display dates in other months (non-selectable) at the start or end of the current month. To make these days selectable use the [selectOtherMonths](#) option.

### Code examples:

Initialize the datepicker with the showOtherMonths option specified:

```
1 | $( ".selector" ).datepicker({ showOtherMonths:  
◀ | _____ ▶
```

Get or set the showOtherMonths option, after initialization:

```
1 | // getter  
2 | var showOtherMonths = $( ".selector" ).datepicker(  
3 |  
4 | // setter  
5 | $( ".selector" ).datepicker( "option", "showOtherMont  
◀ | _____ ▶
```

## showWeek

Type: [Boolean](#)

Default: `false`

When `true`, a column is added to show the week of the year. The [calculateWeek](#) option determines how the week of the year is calculated. You may also want to change the [firstDay](#) option.

### Code examples:

Initialize the datepicker with the showWeek option specified:

```
1 | $( ".selector" ).datepicker({ showWeek: true });  
◀ | _____ ▶
```

Get or set the showWeek option, after initialization:

```
1 // getter
2 var showWeek = $( ".selector" ).datepicker( "option"
3
4 // setter
5 $( ".selector" ).datepicker( "option", "showWeek"
```

## stepMonths

Type: [Number](#)

Default: `1`

Set how many months to move when clicking the previous/next links.

### Code examples:

Initialize the datepicker with the stepMonths option specified:

```
1 $( ".selector" ).datepicker({ stepMonths: 3 });
```

Get or set the stepMonths option, after initialization:

```
1 // getter
2 var stepMonths = $( ".selector" ).datepicker(
3
4 // setter
5 $( ".selector" ).datepicker( "option", "stepMonths"
```

## weekHeader

Type: [String](#)

Default: `"wk"`

The text to display for the week of the year column heading. Use the [showWeek](#) option to display this column.

### Code examples:

Initialize the datepicker with the weekHeader option specified:

```
1 | $( ".selector" ).datepicker({ weekHeader: "W"
```

Get or set the weekHeader option, after initialization:

```
1 | // getter
2 | var weekHeader = $( ".selector" ).datepicker(
3 |
4 | // setter
5 | $( ".selector" ).datepicker( "option", "weekHeader"
```

## yearRange

Type: [String](#)

Default: `"c-10:c+10"`

The range of years displayed in the year drop-down: either relative to today's year (`"-nn:+nn"`), relative to the currently selected year (`"c-nn:c+nn"`), absolute (`"nnnn:nnnn"`), or combinations of these formats (`"nnnn:-nn"`). Note that this option only affects what appears in the drop-down, to restrict which dates may be selected use the [minDate](#) and/or [maxDate](#) options.

### Code examples:

Initialize the datepicker with the yearRange option specified:

```
|
```

```
1 | $( ".selector" ).datepicker({ yearRange: "2002:2012"
```

Get or set the yearRange option, after initialization:

```
1 | // getter
2 | var yearRange = $( ".selector" ).datepicker(
3 |
4 | // setter
5 | $( ".selector" ).datepicker( "option", "yearRange"
```

## yearSuffix

Type: [String](#)

Default: `""`

Additional text to display after the year in the month headers.

### Code examples:

Initialize the datepicker with the yearSuffix option specified:

```
1 | $( ".selector" ).datepicker({ yearSuffix: "CE"
```

Get or set the yearSuffix option, after initialization:

```
1 | // getter
2 | var yearSuffix = $( ".selector" ).datepicker(
3 |
4 | // setter
5 | $( ".selector" ).datepicker( "option", "yearSuffix"
```



## Methods

### destroy()

Removes the datepicker functionality completely. This will return the element back to its pre-init state.

This method does not accept any arguments.

#### Code examples:

Invoke the destroy method:

```
1 | $( ".selector" ).datepicker( "destroy"
```

### dialog( date [, onSelect ] [, settings ] [, pos ] )

Opens the datepicker in a dialog box.

#### date

Type: [String](#) or [Date](#)  
The initial date.

#### onSelect

Type: [Function\(\)](#)  
A callback function when a date is selected. The function receives the date text and date picker instance as parameters.

#### settings

Type: [Options](#)  
The new settings for the date picker.

## pos

Type: `Number[2]` or `MouseEvent`

The position of the top/left of the dialog as `[x, y]` or a `MouseEvent` that contains the coordinates. If not specified the dialog is centered on the screen.

### Code examples:

Invoke the dialog method:

```
1 | $( ".selector" ).datepicker( "dialog", "10/12/2012"
```

## getDate()

Returns: `Date`

Returns the current date for the datepicker or `null` if no date has been selected.

This method does not accept any arguments.

### Code examples:

Invoke the getDate method:

```
1 | var currentDate = $( ".selector" ).datepicker(
```

## hide()

Close a previously opened date picker.

This method does not accept any arguments.

### Code examples:

Invoke the hide method:

```
|
```

```
1 | $( ".selector" ).datepicker( "hide" );
```

## isDisabled()

Returns: [Boolean](#)

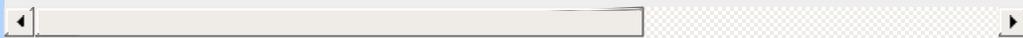
Determine whether a date picker has been disabled.

This method does not accept any arguments.

### Code examples:

Invoke the isDisabled method:

```
1 | var isDisabled = $( ".selector" ).datepicker(
```



## option( optionName )

Returns: [Object](#)

Gets the value currently associated with the specified

`optionName`.

### optionName

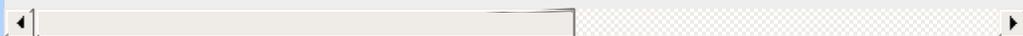
Type: [String](#)

The name of the option to get.

### Code examples:

Invoke the method:

```
1 | var isDisabled = $( ".selector" ).datepicker(
```



## option()

Returns: [PlainObject](#)

Gets an object containing key/value pairs representing the

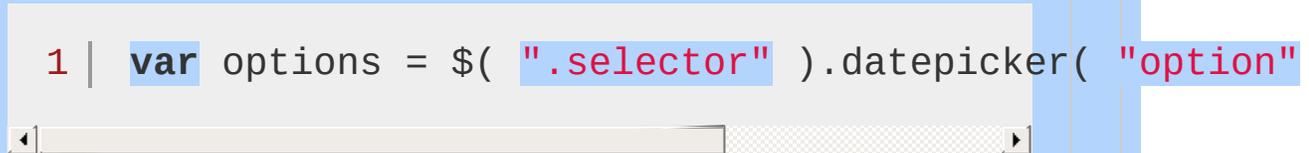
current datepicker options hash.

This method does not accept any arguments.

### Code examples:

Invoke the method:

```
1 | var options = $( ".selector" ).datepicker( "option"
```

A code editor snippet showing a single line of JavaScript code: `1 | var options = $( ".selector" ).datepicker( "option"`. The code is displayed in a light gray box with a scrollbar on the right side.

## option( optionName, value )

Sets the value of the datepicker option associated with the specified `optionName`.

### optionName

Type: String

The name of the option to set.

### value

Type: Object

A value to set for the option.

### Code examples:

Invoke the method:

```
1 | $( ".selector" ).datepicker( "option", "disabled"
```

A code editor snippet showing a single line of JavaScript code: `1 | $( ".selector" ).datepicker( "option", "disabled"`. The code is displayed in a light gray box with a scrollbar on the right side.

## option( options )

Sets one or more options for the datepicker.

### options

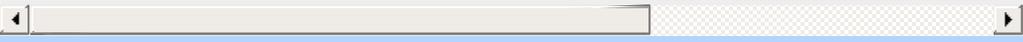
Type: Object

A map of option-value pairs to set.

### Code examples:

Invoke the method:

```
1 | $( ".selector" ).datepicker( "option", { disabled:
```



## refresh()

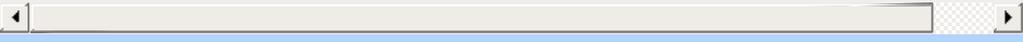
Redraw the date picker, after having made some external modifications.

This method does not accept any arguments.

### Code examples:

Invoke the refresh method:

```
1 | $( ".selector" ).datepicker( "refresh" );
```



## setDate( date )

Sets the date for the datepicker. The new date may be a `Date` object or a string in the current date format (e.g., `"01/26/2009"`), a number of days from today (e.g., `+7`) or a string of values and periods (`"y"` for years, `"m"` for months, `"w"` for weeks, `"d"` for days, e.g., `"+1m +7d"`), or `null` to clear the selected date.

### **date**

Type: String or Date

The new date.

### Code examples:

Invoke the setDate method:

```
1 | $( ".selector" ).datepicker( "setDate", "10/12/2012"
```

## show()

Open the date picker. If the datepicker is attached to an input, the input must be visible for the datepicker to be shown.

This method does not accept any arguments.

### Code examples:

Invoke the show method:

```
1 | $( ".selector" ).datepicker( "show" );
```

## widget()

Returns: *jQuery*

Returns a `jQuery` object containing the datepicker.

This method does not accept any arguments.

### Code examples:

Invoke the widget method:

```
1 | var widget = $( ".selector" ).datepicker( "widget"
```

## Example:

*A simple jQuery UI Datepicker.*

```
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>datepicker demo</title>
6   <link rel="stylesheet" href="http://code.jq
7   <script src="http://code.jquery.com/jquery-:
8   <script src="http://code.jquery.com/ui/1.10
9 </head>
10 <body>
11
12 <div id="datepicker"></div>
13
14 <script>
15 $( "#datepicker" ).datepicker();
16 </script>
17
18 </body>
19 </html>
```

Demo

**A new version of this book is available!**

---

# Dialog Widget

Categories: [Widgets](#)

---

Dialog Widget

version added: 1.0

**Description:** *Open content in an interactive overlay.*

## QuickNav

## Examples

### Options

[appendTo](#)  
[autoOpen](#)  
[buttons](#)  
[closeOnEscape](#)  
[closeText](#)  
[dialogClass](#)  
[draggable](#)  
[height](#)  
[hide](#)  
[maxHeight](#)  
[maxWidth](#)  
[minHeight](#)  
[minWidth](#)  
[modal](#)  
[position](#)  
[resizable](#)  
[show](#)  
[title](#)  
[width](#)

### Methods

[close](#)  
[destroy](#)  
[isOpen](#)  
[moveToTop](#)  
[open](#)  
[option](#)  
[widget](#)

### Events

[beforeClose](#)  
[create](#)  
[open](#)  
[focus](#)  
[dragStart](#)  
[drag](#)  
[dragStop](#)  
[resizeStart](#)  
[resize](#)  
[resizeStop](#)  
[close](#)

A dialog is a floating window that contains a title bar and a content area. The dialog window can be moved, resized and closed with the 'x' icon by default.

If the content length exceeds the maximum height, a scrollbar will automatically appear.

A bottom button bar and semi-transparent modal overlay layer are common options that can be added.

## Hiding the close button

In some cases, you may want to hide the close button, for instance,

if you have a close button in the button pane. The best way to accomplish this is via CSS. As an example, you can define a simple rule, such as:

```
1 | .no-close .ui-dialog-titlebar-close {
2 |     display: none;
3 | }
```

Then, you can simply add the `no-close` class to any dialog in order to hide its close button:

```
1 | $( "#dialog" ).dialog({
2 |     dialogClass: "no-close",
3 |     buttons: [
4 |         {
5 |             text: "OK",
6 |             click: function() {
7 |                 $( this ).dialog( "close" );
8 |             }
9 |         }
10 |     ]
11 | });
```

## Dependencies

[UI Core](#)

[Widget Factory](#)

[Position](#)

[Button](#)

[Draggable](#) (optional; for use with the `draggable` option)

[Resizable](#) (optional; for use with the `resizable` option)

[Effects Core](#) (optional; for use with the `show` and `hide` options)

## Additional Notes:

This widget requires some functional CSS, otherwise it won't work. If you build a custom theme, use the widget's specific CSS file as a starting point.

## Options

### appendTo

Type: [Selector](#)

Default: `"body"`

Which element the dialog (and overlay, if [modal](#)) should be appended to. **(version added: 1.10.0)**

#### Code examples:

Initialize the dialog with the appendTo option specified:

```
1 | $( ".selector" ).dialog({ appendTo: "#sc
```

Get or set the appendTo option, after initialization:

```
1 | // getter
2 | var appendTo = $( ".selector" ).dialog(
3 |
4 | // setter
5 | $( ".selector" ).dialog( "option", "appe
```

### autoOpen

Type: [Boolean](#)

Default: `true`

If set to `true`, the dialog will automatically open upon initialization. If `false`, the dialog will stay hidden until the [open\(\)](#) method is called.

#### Code examples:

Initialize the dialog with the autoOpen option specified:

```
1 |
```

```
1 | $( ".selector" ).dialog({ autoOpen: false });
```

Get or set the autoOpen option, after initialization:

```
1 | // getter
2 | var autoOpen = $( ".selector" ).dialog( "option"
3 |
4 | // setter
5 | $( ".selector" ).dialog( "option", "autoOpen"
```

## buttons

Type: [Object](#) or [Array](#)

Default: `{}`

Specifies which buttons should be displayed on the dialog. The context of the callback is the dialog element; if you need access to the button, it is available as the target of the event object.

### Multiple types supported:

**Object:** The keys are the button labels and the values are the callbacks for when the associated button is clicked.

**Array:** Each element of the array must be an object defining the attributes, properties, and event handlers to set on the button.

### Code examples:

Initialize the dialog with the buttons option specified:

```
1 | $( ".selector" ).dialog({ buttons: [ { text:
```

Get or set the buttons option, after initialization:

```
1 // getter
2 var buttons = $( ".selector" ).dialog( "option"
3
4 // setter
5 $( ".selector" ).dialog( "option", "buttons", [ { tex
```

## closeOnEscape

Type: [Boolean](#)

Default: `true`

Specifies whether the dialog should close when it has focus and the user presses the escape (ESC) key.

### Code examples:

Initialize the dialog with the closeOnEscape option specified:

```
1 $( ".selector" ).dialog({ closeOnEscape: false
```

Get or set the closeOnEscape option, after initialization:

```
1 // getter
2 var closeOnEscape = $( ".selector" ).dialog(
3
4 // setter
5 $( ".selector" ).dialog( "option", "closeOnEscape"
```

## closeText

Type: [String](#)

Default: `"close"`

Specifies the text for the close button. Note that the close text is visibly hidden when using a standard theme.

### Code examples:

Initialize the dialog with the closeText option specified:

```
1 | $( ".selector" ).dialog({ closeText: "hide" });
```

Get or set the closeText option, after initialization:

```
1 | // getter
2 | var closeText = $( ".selector" ).dialog( "option"
3 |
4 | // setter
5 | $( ".selector" ).dialog( "option", "closeText"
```

---

## dialogClass

Type: [String](#)

Default: ""

The specified class name(s) will be added to the dialog, for additional theming.

### Code examples:

Initialize the dialog with the dialogClass option specified:

```
1 | $( ".selector" ).dialog({ dialogClass: "alert"
```

Get or set the dialogClass option, after initialization:

```
1 | // getter
2 | var dialogClass = $( ".selector" ).dialog( "option"
3 |
4 | // setter
5 | $( ".selector" ).dialog( "option", "dialogClass"
```

## draggable

Type: [Boolean](#)

Default: `true`

If set to `true`, the dialog will be draggable by the title bar. Requires the [jQuery UI Draggable widget](#) to be included.

### Code examples:

Initialize the dialog with the draggable option specified:

```
1 | $( ".selector" ).dialog({ draggable: false });
```

Get or set the draggable option, after initialization:

```
1 | // getter
2 | var draggable = $( ".selector" ).dialog( "option"
3 |
4 | // setter
5 | $( ".selector" ).dialog( "option", "draggable"
```

## height

Type: [Number](#) or [String](#)

Default: `"auto"`

The height of the dialog.

### Multiple types supported:

**Number:** The height in pixels.

**String:** The only supported string value is `"auto"` which will allow the dialog height to adjust based on its content.

### Code examples:

Initialize the dialog with the height option specified:

```
1 | $( ".selector" ).dialog({ height: 400 });
```

Get or set the height option, after initialization:

```
1 | // getter
2 | var height = $( ".selector" ).dialog( "option"
3 |
4 | // setter
5 | $( ".selector" ).dialog( "option", "height",
```

**hide**                      **Type:** Number or String or Object

**Default:** `null`

If and how to animate the hiding of the dialog.

**Multiple types supported:**

**Number:** The dialog will fade out while animating the height and width for the specified duration.

**String:** The dialog will be hidden using the specified jQuery UI effect. See the [list of effects](#) for possible values.

**Object:** If the value is an object, then `effect`, `delay`, `duration`, and `easing` properties may be provided. The `effect` property must be the name of a jQuery UI effect. When using a jQuery UI effect that supports additional settings, you may include those settings in the object and they will be passed to the effect. If `duration` or `easing` is omitted, then the default values will be used. If `delay` is omitted, then no delay is used.

**Code examples:**

Initialize the dialog with the hide option specified:

```
1 | $( ".selector" ).dialog({ hide: "explode" });
```

Get or set the hide option, after initialization:

```
1 | // getter
2 | var hide = $( ".selector" ).dialog( "option",
3 |
4 | // setter
5 | $( ".selector" ).dialog( "option", "hide", "explode"
```

## maxHeight

Type: [Number](#)

Default: `false`

The maximum height to which the dialog can be resized, in pixels.

### Code examples:

Initialize the dialog with the maxHeight option specified:

```
1 | $( ".selector" ).dialog({ maxHeight: 600 });
```

Get or set the maxHeight option, after initialization:

```
1 | // getter
2 | var maxHeight = $( ".selector" ).dialog( "option"
3 |
4 | // setter
5 | $( ".selector" ).dialog( "option", "maxHeight"
```

## maxWidth

Type: [Number](#)

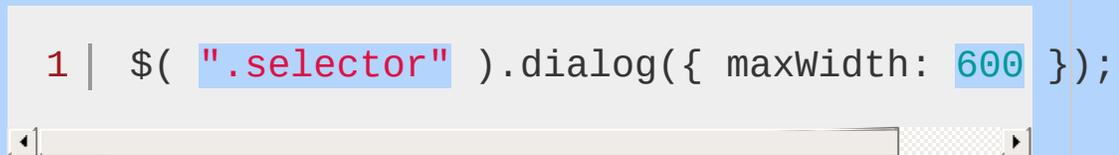
Default: `false`

The maximum width to which the dialog can be resized, in pixels.

### Code examples:

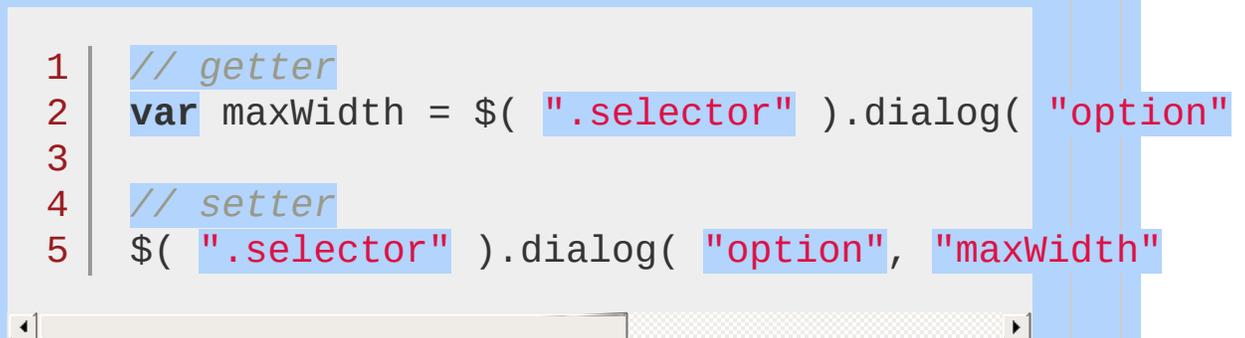
Initialize the dialog with the `maxWidth` option specified:

```
1 | $( ".selector" ).dialog({ maxWidth: 600 });
```



Get or set the `maxWidth` option, after initialization:

```
1 | // getter
2 | var maxWidth = $( ".selector" ).dialog( "option"
3 |
4 | // setter
5 | $( ".selector" ).dialog( "option", "maxWidth"
```



## minHeight

Type: [Number](#)

Default: `150`

The minimum height to which the dialog can be resized, in pixels.

### Code examples:

Initialize the dialog with the `minHeight` option specified:

```
1 | $( ".selector" ).dialog({ minHeight: 200 });
```



Get or set the `minHeight` option, after initialization:

```
1 // getter
2 var minHeight = $( ".selector" ).dialog( "option"
3
4 // setter
5 $( ".selector" ).dialog( "option", "minHeight"
```

## minWidth

Type: [Number](#)

Default: `150`

The minimum width to which the dialog can be resized, in pixels.

### Code examples:

Initialize the dialog with the `minWidth` option specified:

```
1 $( ".selector" ).dialog({ minWidth: 200 });
```

Get or set the `minWidth` option, after initialization:

```
1 // getter
2 var minWidth = $( ".selector" ).dialog( "option"
3
4 // setter
5 $( ".selector" ).dialog( "option", "minWidth"
```

## modal

Type: [Boolean](#)

Default: `false`

If set to `true`, the dialog will have modal behavior; other items on the page will be disabled, i.e., cannot be interacted

with. Modal dialogs create an overlay below the dialog but above other page elements.

### Code examples:

Initialize the dialog with the modal option specified:

```
1 | $( ".selector" ).dialog({ modal: true });
```

Get or set the modal option, after initialization:

```
1 | // getter
2 | var modal = $( ".selector" ).dialog( "option"
3 |
4 | // setter
5 | $( ".selector" ).dialog( "option", "modal", true
```

---

## position

Type: [Object](#) or [String](#) or [Array](#)

Default: `{ my: "center", at: "center", of: window }`

Specifies where the dialog should be displayed. The dialog will handle collisions such that as much of the dialog is visible as possible.

*Note: The [String](#) and [Array](#) forms are deprecated.*

### Multiple types supported:

**Object:** Identifies the position of the dialog when opened. The `of` option defaults to the window, but you can specify another element to position against. You can refer to the [jQuery UI Position](#) utility for more details about the various options.

**String:** A string representing the position within the viewport. Possible values: `"center"`, `"left"`, `"right"`, `"top"`, `"bottom"`.

**Array:** An array containing an  $x, y$  coordinate pair in

pixel offset from the top left corner of the viewport or the name of a possible string value.

### Code examples:

Initialize the dialog with the position option specified:

```
1 | $( ".selector" ).dialog({ position: { my: "left top"
```

Get or set the position option, after initialization:

```
1 | // getter
2 | var position = $( ".selector" ).dialog( "option"
3 |
4 | // setter
5 | $( ".selector" ).dialog( "option", "position"
```

## resizable

Type: [Boolean](#)

Default: `true`

If set to `true`, the dialog will be resizable. Requires the [jQuery UI Resizable widget](#) to be included.

### Code examples:

Initialize the dialog with the resizable option specified:

```
1 | $( ".selector" ).dialog({ resizable: false });
```

Get or set the resizable option, after initialization:

```
1 | // getter
2 | var resizable = $( ".selector" ).dialog( "option"
```

```
3
4 // setter
5 $( ".selector" ).dialog( "option", "resizable"
```

**show**                      **Type:** [Number](#) or [String](#) or [Object](#)

**Default:** `null`

If and how to animate the showing of the dialog.

**Multiple types supported:**

**Number:** The dialog will fade in while animating the height and width for the specified duration.

**String:** The dialog will be shown using the specified jQuery UI effect. See the [list of effects](#) for possible values.

**Object:** If the value is an object, then `effect`, `delay`, `duration`, and `easing` properties may be provided. The `effect` property must be the name of a jQuery UI effect. When using a jQuery UI effect that supports additional settings, you may include those settings in the object and they will be passed to the effect. If `duration` or `easing` is omitted, then the default values will be used. If `delay` is omitted, then no delay is used.

**Code examples:**

Initialize the dialog with the show option specified:

```
1 | $( ".selector" ).dialog({ show: "slow" });
```

Get or set the show option, after initialization:

```
1 | // getter
2 | var show = $( ".selector" ).dialog( "option",
3 |
```

```
4 // setter
5 $( ".selector" ).dialog( "option", "show", "slow"
```

## title

Type: [String](#)

Default: `null`

Specifies the title of the dialog. If the value is `null`, the `title` attribute on the dialog source element will be used.

### Code examples:

Initialize the dialog with the title option specified:

```
1 | $( ".selector" ).dialog({ title: "Dialog Title"
```

Get or set the title option, after initialization:

```
1 // getter
2 var title = $( ".selector" ).dialog( "option"
3
4 // setter
5 $( ".selector" ).dialog( "option", "title", "Dialog T
```

## width

Type: [Number](#)

Default: `300`

The width of the dialog, in pixels.

### Code examples:

Initialize the dialog with the width option specified:

```
1 | $( ".selector" ).dialog({ width: 500 });
```

Get or set the width option, after initialization:

```
1 // getter
2 var width = $( ".selector" ).dialog( "option"
3
4 // setter
5 $( ".selector" ).dialog( "option", "width", 500
```

## Methods

### close()

Closes the dialog.

This method does not accept any arguments.

#### Code examples:

Invoke the close method:

```
1 | $( ".selector" ).dialog( "close" );
```

### destroy()

Removes the dialog functionality completely. This will return the element back to its pre-init state.

This method does not accept any arguments.

#### Code examples:

Invoke the destroy method:

```
1 | $( ".selector" ).dialog( "destroy" );
```

### isOpen()

Returns: [Boolean](#)

Whether the dialog is currently open.

This method does not accept any arguments.

### Code examples:

Invoke the isOpen method:

```
1 | var isOpen = $( ".selector" ).dialog( "isOpen"
```

## moveToTop()

Moves the dialog to the top of the dialog stack.

This method does not accept any arguments.

### Code examples:

Invoke the moveToTop method:

```
1 | $( ".selector" ).dialog( "moveToTop" );
```

## open()

Opens the dialog.

This method does not accept any arguments.

### Code examples:

Invoke the open method:

```
1 | $( ".selector" ).dialog( "open" );
```

**option( optionName )**

Returns: *Object*

Gets the value currently associated with the specified `optionName`.

**optionName**

Type: `String`

The name of the option to get.

### Code examples:

Invoke the method:

```
1 | var isDisabled = $( ".selector" ).dialog( "option"
```

## option()

Returns: `PlainObject`

Gets an object containing key/value pairs representing the current dialog options hash.

This method does not accept any arguments.

### Code examples:

Invoke the method:

```
1 | var options = $( ".selector" ).dialog( "option"
```

## option( optionName, value )

Sets the value of the dialog option associated with the specified `optionName`.

**optionName**

Type: `String`

The name of the option to set.

## value

Type: [Object](#)

A value to set for the option.

### Code examples:

Invoke the method:

```
1 | $( ".selector" ).dialog( "option", "disabled"
```

## option( options )

Sets one or more options for the dialog.

### options

Type: [Object](#)

A map of option-value pairs to set.

### Code examples:

Invoke the method:

```
1 | $( ".selector" ).dialog( "option", { disabled:
```

## widget()

Returns: [jQuery](#)

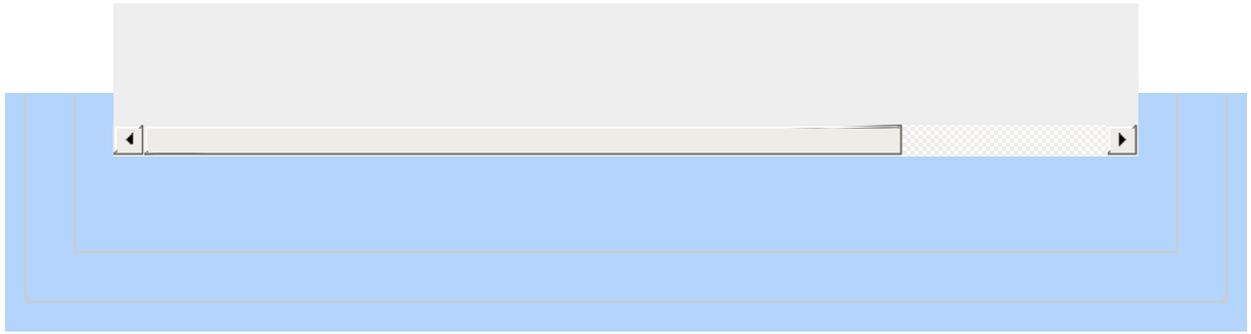
Returns a [jQuery](#) object containing the generated wrapper.

This method does not accept any arguments.

### Code examples:

Invoke the widget method:

```
1 | var widget = $( ".selector" ).dialog( "widget"
```



## Events

**beforeClose( event, ui )** *Type:* `dialogbeforeclose`

Triggered when a dialog is about to close. If canceled, the dialog will not close.

**event**  
Type: [Event](#)

**ui**  
Type: [Object](#)

### Code examples:

Initialize the dialog with the beforeClose callback specified:

```
1 | $( ".selector" ).dialog({  
2 |   beforeClose: function( event, ui ) {}  
3 | });
```

Bind an event listener to the dialogbeforeclose event:

```
1 | $( ".selector" ).on( "dialogbeforeclose"
```

**close( event, ui )** *Type:* `dialogclose`

Triggered when the dialog is closed.

**event**  
Type: [Event](#)

**ui**

Type: [Object](#)

### Code examples:

Initialize the dialog with the close callback specified:

```
1 | $( ".selector" ).dialog({
2 |   close: function( event, ui ) {}
3 | });
```

Bind an event listener to the dialogclose event:

```
1 | $( ".selector" ).on( "dialogclose", function( event,
```



**create( event, ui )**

Type: [dialogcreate](#)

Triggered when the dialog is created.

**event**

Type: [Event](#)

**ui**

Type: [Object](#)

### Code examples:

Initialize the dialog with the create callback specified:

```
1 | $( ".selector" ).dialog({
2 |   create: function( event, ui ) {}
3 | });
```

Bind an event listener to the dialogcreate event:

```
1 | $( ".selector" ).on( "dialogcreate", function
```

## drag( event, ui )

Type: `dialogdrag`

Triggered while the dialog is being dragged.

### event

Type: `Event`

### ui

Type: `Object`

#### position

Type: `Object`

The current CSS position of the dialog.

#### offset

Type: `Object`

The current offset position of the dialog.

### Code examples:

Initialize the dialog with the drag callback specified:

```
1 | $( ".selector" ).dialog({  
2 |   drag: function( event, ui ) {}  
3 | });
```

Bind an event listener to the dialogdrag event:

```
1 | $( ".selector" ).on( "dialogdrag", function( event, ui
```

## dragStart( event, ui ) *Type:* dialogdragstart

Triggered when the user starts dragging the dialog.

### event

Type: [Event](#)

### ui

Type: [Object](#)

#### position

Type: [Object](#)

The current CSS position of the dialog.

#### offset

Type: [Object](#)

The current offset position of the dialog.

### Code examples:

Initialize the dialog with the dragStart callback specified:

```
1 | $( ".selector" ).dialog({  
2 |   dragStart: function( event, ui ) {}  
3 | });
```

Bind an event listener to the dialogdragstart event:

```
1 | $( ".selector" ).on( "dialogdragstart", function
```



---

## dragStop( event, ui ) *Type:* dialogdragstop

Triggered after the dialog has been dragged.

### event

Type: [Event](#)

**ui**

Type: [Object](#)

**position**

Type: [Object](#)

The current CSS position of the dialog.

**offset**

Type: [Object](#)

The current offset position of the dialog.

### Code examples:

Initialize the dialog with the dragStop callback specified:

```
1 | $( ".selector" ).dialog({  
2 |   dragStop: function( event, ui ) {}  
3 | });
```

Bind an event listener to the dialogdragstop event:

```
1 | $( ".selector" ).on( "dialogdragstop", function
```



---

**focus( event, ui )**

Type: [dialogfocus](#)

Triggered when the dialog gains focus.

**event**

Type: [Event](#)

**ui**

Type: [Object](#)

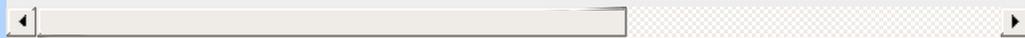
### Code examples:

Initialize the dialog with the focus callback specified:

```
1 | $( ".selector" ).dialog({  
2 |   focus: function( event, ui ) {}  
3 | });
```

Bind an event listener to the dialogfocus event:

```
1 | $( ".selector" ).on( "dialogfocus", function( event,
```



---

### open( event, ui )

Type: `dialogopen`

Triggered when the dialog is opened.

**event**

Type: [Event](#)

**ui**

Type: [Object](#)

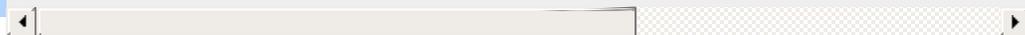
### Code examples:

Initialize the dialog with the open callback specified:

```
1 | $( ".selector" ).dialog({  
2 |   open: function( event, ui ) {}  
3 | });
```

Bind an event listener to the dialogopen event:

```
1 | $( ".selector" ).on( "dialogopen", function( event, u
```



## resize( event, ui )

Type: `dialogresize`

Triggered while the dialog is being resized.

### event

Type: [Event](#)

### ui

Type: [Object](#)

#### originalPosition

Type: [Object](#)

The CSS position of the dialog prior to being resized.

#### position

Type: [Object](#)

The current CSS position of the dialog.

#### originalSize

Type: [Object](#)

The size of the dialog prior to being resized.

#### size

Type: [Object](#)

The current size of the dialog.

### Code examples:

Initialize the dialog with the resize callback specified:

```
1 | $( ".selector" ).dialog({  
2 |   resize: function( event, ui ) {}  
3 | });
```

Bind an event listener to the dialogresize event:

```
1 | $( ".selector" ).on( "dialogresize", function
```

**resizeStart( event, ui )** *Type:* `dialogresizestart`

Triggered when the user starts resizing the dialog.

**event**

Type: [Event](#)

**ui**

Type: [Object](#)

**originalPosition**

Type: [Object](#)

The CSS position of the dialog prior to being resized.

**position**

Type: [Object](#)

The current CSS position of the dialog.

**originalSize**

Type: [Object](#)

The size of the dialog prior to being resized.

**size**

Type: [Object](#)

The current size of the dialog.

### Code examples:

Initialize the dialog with the `resizeStart` callback specified:

```
1 | $( ".selector" ).dialog({  
2 |   resizeStart: function( event, ui ) {}  
3 | });
```

Bind an event listener to the dialogresizestart event:

```
1 | $( ".selector" ).on( "dialogresizestart", function
```

**resizeStop( event, ui )** *Type:* dialogresizestop

Triggered after the dialog has been resized.

**event**

Type: [Event](#)

**ui**

Type: [Object](#)

**originalPosition**

Type: [Object](#)

The CSS position of the dialog prior to being resized.

**position**

Type: [Object](#)

The current CSS position of the dialog.

**originalSize**

Type: [Object](#)

The size of the dialog prior to being resized.

**size**

Type: [Object](#)

The current size of the dialog.

**Code examples:**

Initialize the dialog with the resizeStop callback specified:

```
1 | $( ".selector" ).dialog({  
2 |     resizeStop: function( event, ui ) {}  
3 | });
```

Bind an event listener to the dialogresizestop event:

```
1 | $( ".selector" ).on( "dialogresizestop", function
```



## Example:

### *A simple jQuery UI Dialog*

```
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>dialog demo</title>
6   <link rel="stylesheet" href="http://code.jq
7   <script src="http://code.jquery.com/jquery-:
8   <script src="http://code.jquery.com/ui/1.10
9 </head>
10 <body>
11
12 <button id="opener">open the dialog</button>
13 <div id="dialog" title="Dialog Title">I'm a d.
14
15 <script>
16 $( "#dialog" ).dialog({ autoOpen: false });
17 $( "#opener" ).click(function() {
18   $( "#dialog" ).dialog( "open" );
19 });
20 </script>
21
22 </body>
23 </html>
```



Demo

**A new version of this book is available!**

---

# .disableSelection()

Categories: [Methods](#) | [UI Core](#)

`.disableSelection()`

Returns: *jQuery*

**Description:** *Disable selection of text content within the set of matched elements.*

**.disableSelection()**

version added: 1.6, deprecated: 1.9

This method does not accept any arguments.

Disabling text selection is bad. Don't use this.

POWERED BYHERONOTE

A new version of this book is available!

---

# Draggable Widget

Categories: [Interactions](#)

---

**Draggable Widget**

version added: 1.0

**Description:** *Allow elements to be moved using the mouse.*

## QuickNav

## Examples

### Options

[addClasses](#)  
[appendTo](#)  
[axis](#)  
[cancel](#)  
[connectToSortable](#)  
[containment](#)  
[cursor](#)  
[cursorAt](#)  
[delay](#)  
[disabled](#)  
[distance](#)  
[grid](#)  
[handle](#)  
[helper](#)  
[iframeFix](#)  
[opacity](#)  
[refreshPositions](#)  
[revert](#)  
[revertDuration](#)  
[scope](#)  
[scroll](#)  
[scrollSensitivity](#)  
[scrollSpeed](#)  
[snap](#)  
[snapMode](#)  
[snapTolerance](#)  
[stack](#)  
[zIndex](#)

### Methods

[destroy](#)  
[disable](#)  
[enable](#)  
[option](#)  
[widget](#)

### Events

[create](#)  
[start](#)  
[drag](#)  
[stop](#)

Make the selected elements draggable by mouse. If you want not just drag, but drag & drop, see the [jQuery UI Droppable plugin](#), which provides a drop target for draggables.

## Dependencies

[UI Core](#)

[Widget Factory](#)

[Mouse Interaction](#)

## Options

### addClasses

Type: [Boolean](#)

Default: `true`

If set to `false`, will prevent the `ui-draggable` class from being added. This may be desired as a performance optimization when calling `.draggable()` on hundreds of elements.

#### Code examples:

Initialize the draggable with the addClasses option specified:

```
1 | $( ".selector" ).draggable({ addClasses
```

Get or set the addClasses option, after initialization:

```
1 | // getter
2 | var addClasses = $( ".selector" ).dragg
3 |
4 | // setter
5 | $( ".selector" ).draggable( "option", "a
```

### appendTo

Default: Type: [jQuery](#) or [Element](#) or [Selector](#) or [String](#)  
`"parent"`

Which element the draggable helper should be appended to while dragging.

#### Multiple types supported:

**jQuery:** A jQuery object containing the element to

append the helper to.

**Element:** The element to append the helper to.

**Selector:** A selector specifying which element to append the helper to.

**String:** The string "parent" will cause the helper to be a sibling of the draggable.

### Code examples:

Initialize the draggable with the appendTo option specified:

```
1 | $( ".selector" ).draggable({ appendTo: "body"
```

Get or set the appendTo option, after initialization:

```
1 | // getter
2 | var appendTo = $( ".selector" ).draggable( "option"
3 |
4 | // setter
5 | $( ".selector" ).draggable( "option", "appendTo"
```

## axis

**Type:** String

**Default:** false

Constrains dragging to either the horizontal (x) or vertical (y) axis. Possible values: "x", "y".

### Code examples:

Initialize the draggable with the axis option specified:

```
1 | $( ".selector" ).draggable({ axis: "x" });
```

Get or set the axis option, after initialization:

```
1 | // getter
2 | var axis = $( ".selector" ).draggable( "option"
3 |
4 | // setter
5 | $( ".selector" ).draggable( "option", "axis",
```

## cancel

Type: [Selector](#)

Default: `"input, textarea, button, select, option"`

Prevents dragging from starting on specified elements.

### Code examples:

Initialize the draggable with the cancel option specified:

```
1 | $( ".selector" ).draggable({ cancel: ".title"
```

Get or set the cancel option, after initialization:

```
1 | // getter
2 | var cancel = $( ".selector" ).draggable( "option"
3 |
4 | // setter
5 | $( ".selector" ).draggable( "option", "cancel"
```

## connectToSortable

Type: [Selector](#)

Default: `false`

Allows the draggable to be dropped onto the specified

sortable. If this option is used, a draggable can be dropped onto a sortable list and then becomes part of it. Note: The `helper` option must be set to `"clone"` in order to work flawlessly. Requires the [jQuery UI Sortable plugin](#) to be included.

### Code examples:

Initialize the draggable with the `connectToSortable` option specified:

```
1 | $( ".selector" ).draggable({ connectToSortable:
```

Get or set the `connectToSortable` option, after initialization:

```
1 | // getter
2 | var connectToSortable = $( ".selector" ).draggable(
3 |
4 | // setter
5 | $( ".selector" ).draggable( "option", "connectToSortable"
```

## containment

Default: **Type:** [Selector](#) or [Element](#) or [String](#) or [Array](#)  
`false`

Constrains dragging to within the bounds of the specified element or region.

### Multiple types supported:

**Selector:** The draggable element will be contained to the bounding box of the first element found by the selector. If no element is found, no containment will be set.

**Element:** The draggable element will be contained to the bounding box of this element.

**String:** Possible values: `"parent"`, `"document"`,

```
"window".
```

**Array:** An array defining a bounding box in the form [ x1, y1, x2, y2 ].

### Code examples:

Initialize the draggable with the containment option specified:

```
1 | $( ".selector" ).draggable({ containment: "parent"
```

Get or set the containment option, after initialization:

```
1 | // getter
2 | var containment = $( ".selector" ).draggable(
3 |
4 | // setter
5 | $( ".selector" ).draggable( "option", "containment"
```

## cursor

Type: String

Default: "auto"

The CSS cursor during the drag operation.

### Code examples:

Initialize the draggable with the cursor option specified:

```
1 | $( ".selector" ).draggable({ cursor: "crosshair"
```

Get or set the cursor option, after initialization:

```
1 | // getter
```

```
2 | var cursor = $( ".selector" ).draggable( "option"  
3 |  
4 | // setter  
5 | $( ".selector" ).draggable( "option", "cursor"
```

## cursorAt

Type: [Object](#)

Default: `false`

Sets the offset of the dragging helper relative to the mouse cursor. Coordinates can be given as a hash using a combination of one or two keys: `{ top, left, right, bottom }`.

### Code examples:

Initialize the draggable with the cursorAt option specified:

```
1 | $( ".selector" ).draggable({ cursorAt: { left:
```

Get or set the cursorAt option, after initialization:

```
1 | // getter  
2 | var cursorAt = $( ".selector" ).draggable( "option"  
3 |  
4 | // setter  
5 | $( ".selector" ).draggable( "option", "cursorAt"
```

## delay

Type: [Number](#)

Default: `0`

Time in milliseconds after mousedown until dragging should start. This option can be used to prevent unwanted drags

when clicking on an element.

### Code examples:

Initialize the draggable with the delay option specified:

```
1 | $( ".selector" ).draggable({ delay: 300 });
```

Get or set the delay option, after initialization:

```
1 | // getter
2 | var delay = $( ".selector" ).draggable( "option"
3 |
4 | // setter
5 | $( ".selector" ).draggable( "option", "delay"
```

## disabled

Type: Boolean

Default: `false`

Disables the draggable if set to `true`.

### Code examples:

Initialize the draggable with the disabled option specified:

```
1 | $( ".selector" ).draggable({ disabled: true });
```

Get or set the disabled option, after initialization:

```
1 | // getter
2 | var disabled = $( ".selector" ).draggable( "option"
3 |
4 | // setter
5 | $( ".selector" ).draggable( "option", "disabled"
```

## distance

Type: [Number](#)

Default: `1`

Distance in pixels after mousedown the mouse must move before dragging should start. This option can be used to prevent unwanted drags when clicking on an element.

### Code examples:

Initialize the draggable with the distance option specified:

```
1 | $( ".selector" ).draggable({ distance: 10 });
```

Get or set the distance option, after initialization:

```
1 | // getter
2 | var distance = $( ".selector" ).draggable( "option"
3 |
4 | // setter
5 | $( ".selector" ).draggable( "option", "distance"
```

## grid

Type: [Array](#)

Default: `false`

Snaps the dragging helper to a grid, every x and y pixels. The array must be of the form `[ x, y ]`.

### Code examples:

Initialize the draggable with the grid option specified:

```
1 | $( ".selector" ).draggable({ grid: [ 50, 20 ] });
```

Get or set the grid option, after initialization:

```
1 // getter
2 var grid = $( ".selector" ).draggable( "option"
3
4 // setter
5 $( ".selector" ).draggable( "option", "grid", [
```

**handle** Type: [Selector](#) or [Element](#)

Default: `false`

If specified, restricts dragging from starting unless the mousedown occurs on the specified element(s).

**Code examples:**

Initialize the draggable with the handle option specified:

```
1 $( ".selector" ).draggable({ handle: "h2" });
```

Get or set the handle option, after initialization:

```
1 // getter
2 var handle = $( ".selector" ).draggable( "option"
3
4 // setter
5 $( ".selector" ).draggable( "option", "handle"
```

## helper

Type: [String](#) or [Function\(\)](#)

Default: `"original"`

Allows for a helper element to be used for dragging display.

### Multiple types supported:

**String:** If set to `"clone"`, then the element will be cloned and the clone will be dragged.

**Function:** A function that will return a `DOMElement` to use while dragging.

### Code examples:

Initialize the draggable with the helper option specified:

```
1 | $( ".selector" ).draggable({ helper: "clone" });
```

Get or set the helper option, after initialization:

```
1 | // getter
2 | var helper = $( ".selector" ).draggable( "option"
3 |
4 | // setter
5 | $( ".selector" ).draggable( "option", "helper"
```

## iframeFix

Type: [Boolean](#) or [Selector](#)

Default: `false`

Prevent iframes from capturing the mousemove events during a drag. Useful in combination with the [cursorAt](#) option, or in any case where the mouse cursor may not be over the helper.

### Multiple types supported:

**Boolean:** When set to `true`, transparent overlays will

be placed over all iframes on the page.

**Selector:** Any iframes matching the selector will be covered by transparent overlays.

### Code examples:

Initialize the draggable with the `iframeFix` option specified:

```
1 | $( ".selector" ).draggable({ iframeFix: true });
```

Get or set the `iframeFix` option, after initialization:

```
1 | // getter
2 | var iframeFix = $( ".selector" ).draggable( "option"
3 |
4 | // setter
5 | $( ".selector" ).draggable( "option", "iframeFix"
```

## opacity

Type: [Number](#)

Default: `false`

Opacity for the helper while being dragged.

### Code examples:

Initialize the draggable with the `opacity` option specified:

```
1 | $( ".selector" ).draggable({ opacity: 0.35 });
```

Get or set the `opacity` option, after initialization:

```
1 | // getter
2 | var opacity = $( ".selector" ).draggable( "option"
```

```
3
4 // setter
5 $( ".selector" ).draggable( "option", "opacity"
```

## refreshPositions

Type: [Boolean](#)

Default: `false`

If set to `true`, all droppable positions are calculated on every `mousemove`. *Caution: This solves issues on highly dynamic pages, but dramatically decreases performance.*

### Code examples:

Initialize the draggable with the refreshPositions option specified:

```
1 | $( ".selector" ).draggable({ refreshPositions:
```

Get or set the refreshPositions option, after initialization:

```
1 | // getter
2 | var refreshPositions = $( ".selector" ).draggable(
3 |
4 | // setter
5 | $( ".selector" ).draggable( "option", "refreshPositi
```

## revert

Type: [Boolean](#) or [String](#)

Default: `false`

Whether the element should revert to its start position when dragging stops.

Multiple types supported:

**Boolean:** If set to `true` the element will always revert.

**String:** If set to `"invalid"`, revert will only occur if the draggable has not been dropped on a droppable. For `"valid"`, it's the other way around.

### Code examples:

Initialize the draggable with the revert option specified:

```
1 | $( ".selector" ).draggable({ revert: true });
```

Get or set the revert option, after initialization:

```
1 | // getter
2 | var revert = $( ".selector" ).draggable( "option"
3 |
4 | // setter
5 | $( ".selector" ).draggable( "option", "revert"
```

## revertDuration

Type: [Number](#)

Default: `500`

The duration of the revert animation, in milliseconds. Ignored if the `revert` option is `false`.

### Code examples:

Initialize the draggable with the revertDuration option specified:

```
1 | $( ".selector" ).draggable({ revertDuration:
```

Get or set the revertDuration option, after initialization:

```
1 // getter
2 var revertDuration = $( ".selector" ).draggable(
3
4 // setter
5 $( ".selector" ).draggable( "option", "revertDuration"
```

## scope

Type: [String](#)

Default: `"default"`

Used to group sets of draggable and droppable items, in addition to droppable's `accept` option. A draggable with the same `scope` value as a droppable will be accepted by the droppable.

### Code examples:

Initialize the draggable with the scope option specified:

```
1 $( ".selector" ).draggable({ scope: "tasks" });
```

Get or set the scope option, after initialization:

```
1 // getter
2 var scope = $( ".selector" ).draggable( "option"
3
4 // setter
5 $( ".selector" ).draggable( "option", "scope"
```

## scroll

Type: [Boolean](#)

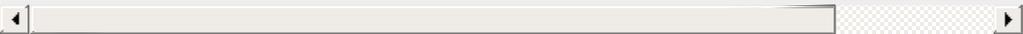
Default: `true`

If set to `true`, container auto-scrolls while dragging.

### Code examples:

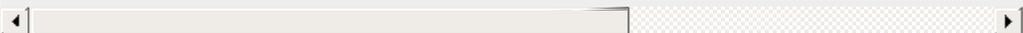
Initialize the draggable with the scroll option specified:

```
1 | $( ".selector" ).draggable({ scroll: false });
```



Get or set the scroll option, after initialization:

```
1 | // getter
2 | var scroll = $( ".selector" ).draggable( "option"
3 |
4 | // setter
5 | $( ".selector" ).draggable( "option", "scroll"
```



## scrollSensitivity

Type: [Number](#)

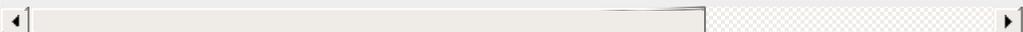
Default: `20`

Distance in pixels from the edge of the viewport after which the viewport should scroll. Distance is relative to pointer, not the draggable. Ignored if the `scroll` option is `false`.

### Code examples:

Initialize the draggable with the scrollSensitivity option specified:

```
1 | $( ".selector" ).draggable({ scrollSensitivity:
```



Get or set the scrollSensitivity option, after initialization:

```
1 | // getter
2 | var scrollSensitivity = $( ".selector" ).draggable(
```

```
3
4 // setter
5 $( ".selector" ).draggable( "option", "scrollSensitivity"
```

## scrollTop

Type: [Number](#)

Default: `20`

The speed at which the window should scroll once the mouse pointer gets within the `scrollSensitivity` distance. Ignored if the `scroll` option is `false`.

### Code examples:

Initialize the draggable with the scrollTop option specified:

```
1 | $( ".selector" ).draggable({ scrollTop: 100
```

Get or set the scrollTop option, after initialization:

```
1 // getter
2 var scrollTop = $( ".selector" ).draggable(
3
4 // setter
5 $( ".selector" ).draggable( "option", "scrollTop"
```

## snap

Type: [Boolean](#) or [Selector](#)

Default: `false`

Whether the element should snap to other elements.

### Multiple types supported:

**Boolean:** When set to `true`, the element will snap to all other draggable elements.

**Selector:** A selector specifying which elements to snap to.

### Code examples:

Initialize the draggable with the snap option specified:

```
1 | $( ".selector" ).draggable({ snap: true });
```

Get or set the snap option, after initialization:

```
1 | // getter
2 | var snap = $( ".selector" ).draggable( "option"
3 |
4 | // setter
5 | $( ".selector" ).draggable( "option", "snap",
```

## snapMode

Type: String

Default: "both"

Determines which edges of snap elements the draggable will snap to. Ignored if the `snap` option is `false`. Possible values: "inner", "outer", "both".

### Code examples:

Initialize the draggable with the snapMode option specified:

```
1 | $( ".selector" ).draggable({ snapMode: "inner"
```

Get or set the snapMode option, after initialization:

```
1 | // getter
```

```
2 var snapMode = $( ".selector" ).draggable( "option"  
3  
4 // setter  
5 $( ".selector" ).draggable( "option", "snapMode"
```

## snapTolerance

Type: [Number](#)

Default: `20`

The distance in pixels from the snap element edges at which snapping should occur. Ignored if the `snap` option is `false`.

### Code examples:

Initialize the draggable with the `snapTolerance` option specified:

```
1 | $( ".selector" ).draggable({ snapTolerance: 30
```

Get or set the `snapTolerance` option, after initialization:

```
1 // getter  
2 var snapTolerance = $( ".selector" ).draggable(  
3  
4 // setter  
5 $( ".selector" ).draggable( "option", "snapTolerance"
```

## stack

Type: [Selector](#)

Default: `false`

Controls the z-index of the set of elements that match the selector, always brings the currently dragged item to the front. Very useful in things like window managers.

## Code examples:

Initialize the draggable with the stack option specified:

```
1 | $( ".selector" ).draggable({ stack: ".products"
```

Get or set the stack option, after initialization:

```
1 | // getter
2 | var stack = $( ".selector" ).draggable( "option"
3 |
4 | // setter
5 | $( ".selector" ).draggable( "option", "stack"
```

## zIndex

Type: Number

Default: `false`

Z-index for the helper while being dragged.

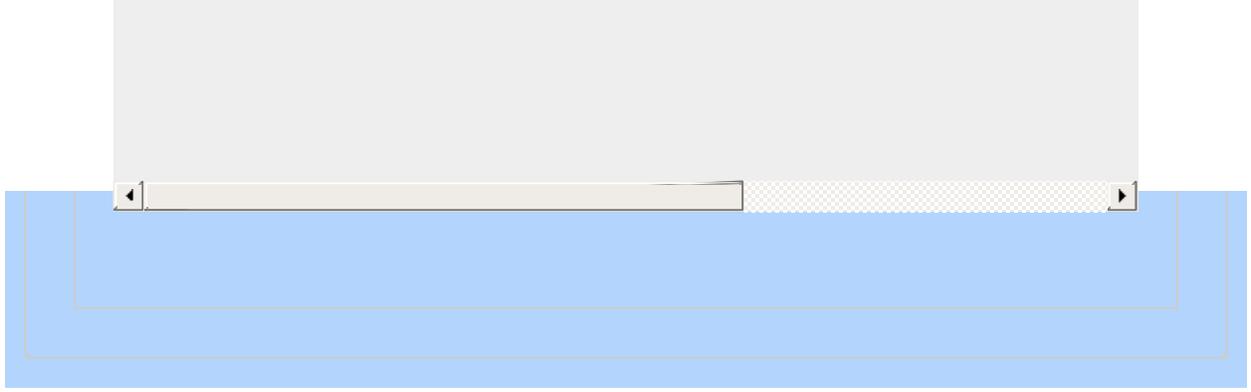
## Code examples:

Initialize the draggable with the zIndex option specified:

```
1 | $( ".selector" ).draggable({ zIndex: 100 });
```

Get or set the zIndex option, after initialization:

```
1 | // getter
2 | var zIndex = $( ".selector" ).draggable( "option"
3 |
4 | // setter
5 | $( ".selector" ).draggable( "option", "zIndex"
```



## Methods

### destroy()

Removes the draggable functionality completely. This will return the element back to its pre-init state.

This method does not accept any arguments.

#### Code examples:

Invoke the destroy method:

```
1 | $( ".selector" ).draggable( "destroy" );
```

### disable()

Disables the draggable.

This method does not accept any arguments.

#### Code examples:

Invoke the disable method:

```
1 | $( ".selector" ).draggable( "disable" );
```

### enable()

Enables the draggable.

This method does not accept any arguments.

### Code examples:

Invoke the enable method:

```
1 | $( ".selector" ).draggable( "enable" );
```

## option( optionName )

Returns: *Object*

Gets the value currently associated with the specified `optionName`.

**optionName**

Type: *String*

The name of the option to get.

### Code examples:

Invoke the method:

```
1 | var isDisabled = $( ".selector" ).draggable(
```

## option()

Returns: *PlainObject*

Gets an object containing key/value pairs representing the current draggable options hash.

This method does not accept any arguments.

### Code examples:

Invoke the method:

```
1 | var options = $( ".selector" ).draggable( "option"
```

## option( optionName, value )

Sets the value of the draggable option associated with the specified `optionName`.

### **optionName**

Type: String

The name of the option to set.

### **value**

Type: Object

A value to set for the option.

### **Code examples:**

Invoke the method:

```
1 | $( ".selector" ).draggable( "option", "disabled"
```

## option( options )

Sets one or more options for the draggable.

### **options**

Type: Object

A map of option-value pairs to set.

### **Code examples:**

Invoke the method:

```
1 | $( ".selector" ).draggable( "option", { disabled:
```

## widget()

Returns: *jQuery*

Returns a `jQuery` object containing the draggable element.

This method does not accept any arguments.

### Code examples:

Invoke the `widget` method:

```
1 | var widget = $( ".selector" ).draggable( "widget"
```

## Events

**create( event, ui )**

Type: `dragcreate`

Triggered when the draggable is created.

**event**

Type: `Event`

**ui**

Type: `Object`

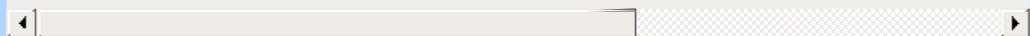
### Code examples:

Initialize the draggable with the create callback specified:

```
1 | $( ".selector" ).draggable({
2 |   create: function( event, ui ) {}
3 | });
```

Bind an event listener to the dragcreate event:

```
1 | $( ".selector" ).on( "dragcreate", funct
```



**drag( event, ui )**

Type: `drag`

Triggered while the mouse is moved during the dragging.

**event**

Type: `Event`

**ui**

Type: Object

### helper

Type: jQuery

The jQuery object representing the helper that's being dragged.

### position

Type: Object

Current CSS position of the helper as `{ top, left }` object.

### offset

Type: Object

Current offset position of the helper as `{ top, left }` object.

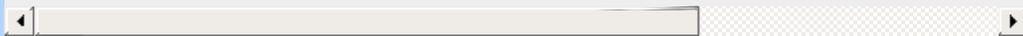
## Code examples:

Initialize the draggable with the drag callback specified:

```
1 | $( ".selector" ).draggable({  
2 |   drag: function( event, ui ) {}  
3 | });
```

Bind an event listener to the drag event:

```
1 | $( ".selector" ).on( "drag", function( event, ui ) {
```



---

## start( event, ui )

Type: `dragstart`

Triggered when dragging starts.

### event

Type: Event

## ui

Type: `Object`

### helper

Type: `jQuery`

The jQuery object representing the helper that's being dragged.

### position

Type: `Object`

Current CSS position of the helper as `{ top, left }` object.

### offset

Type: `Object`

Current offset position of the helper as `{ top, left }` object.

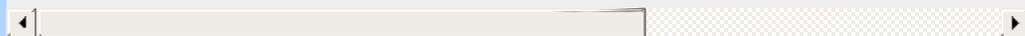
## Code examples:

Initialize the draggable with the start callback specified:

```
1 | $( ".selector" ).draggable({  
2 |   start: function( event, ui ) {}  
3 | });
```

Bind an event listener to the dragstart event:

```
1 | $( ".selector" ).on( "dragstart", function( event, ui
```



## stop( event, ui )

Type: `dragstop`

Triggered when dragging stops.

**event**

Type: [Event](#)

**ui**

Type: [Object](#)

**helper**

Type: [jQuery](#)

The jQuery object representing the helper that's being dragged.

**position**

Type: [Object](#)

Current CSS position of the helper as `{ top, left }` object.

**offset**

Type: [Object](#)

Current offset position of the helper as `{ top, left }` object.

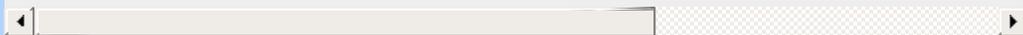
### Code examples:

Initialize the draggable with the stop callback specified:

```
1 | $( ".selector" ).draggable({  
2 |   stop: function( event, ui ) {}  
3 | });
```

Bind an event listener to the dragstop event:

```
1 | $( ".selector" ).on( "dragstop", function( event, ui
```



## Example:

### *A simple jQuery UI Draggable*

```
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>draggable demo</title>
6   <link rel="stylesheet" href="http://code.jq
7   <style>
8     #draggable {
9       width: 100px;
10      height: 100px;
11      background: #ccc;
12    }
13  </style>
14  <script src="http://code.jquery.com/jquery-:
15  <script src="http://code.jquery.com/ui/1.10
16 </head>
17 <body>
18
19 <div id="draggable">Drag me</div>
20
21 <script>
22 $( "#draggable" ).draggable();
23 </script>
24
25 </body>
26 </html>
```

Demo

POWERED BYHERONOTE

A new version of this book is available!

---

# Drop Effect

Categories: [Effects](#)

## Drop Effect

**Description:** *The drop effect hides or shows an element fading in/out and sliding in a direction.*

### drop

**direction** (default: `"left"`)

Type: String

The direction the element will fall to hide the element, or the direction from which the element will be revealed.

Possible Values: `up`, `down`, `left`, `right`.

## Example:

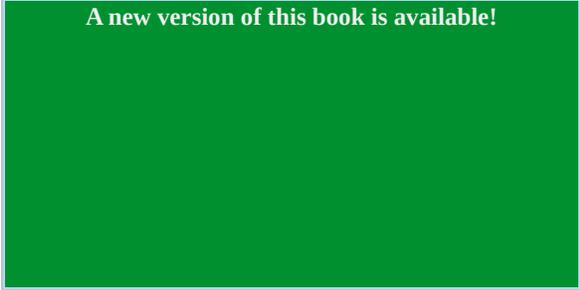
*Toggle a div using the drop effect.*

```
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>drop demo</title>
6   <link rel="stylesheet" href="http://code.jq
7   <style>
8     #toggle {
9       width: 100px;
10      height: 100px;
11      background: #ccc;
12    }
13  </style>
14  <script src="http://code.jquery.com/jquery-:
15  <script src="http://code.jquery.com/ui/1.10
16 </head>
17 <body>
18
19 <p>Click anywhere to toggle the box.</p>
20 <div id="toggle"></div>
21
22 <script>
23 $( document ).click(function() {
24   $( "#toggle" ).toggle( "drop" );
25 });
26 </script>
27
28 </body>
29 </html>
```

Demo

POWERED BYHERONOTE

A new version of this book is available!



---

# Droppable Widget

Categories: [Interactions](#)

---

**Droppable Widget**

version added: 1.0

**Description:** *Create targets for draggable elements.*

## QuickNav

## Examples

### Options

[accept](#)  
[activeClass](#)  
[addClasses](#)  
[disabled](#)  
[greedy](#)  
[hoverClass](#)  
[scope](#)  
[tolerance](#)

### Methods

[destroy](#)  
[disable](#)  
[enable](#)  
[option](#)  
[widget](#)

### Events

[create](#)  
[activate](#)  
[deactivate](#)  
[over](#)  
[out](#)  
[drop](#)

The jQuery UI Droppable plugin makes selected elements droppable (meaning they accept being dropped on by [draggables](#)). You can specify which draggables each will accept.

## Dependencies

[UI Core](#)

[Widget Factory](#)

[Mouse Interaction](#)

## Options

**accept**

Type: [Selector](#) or [Function\(\)](#)

Default: `"*"`

Controls which draggable elements are accepted by the droppable.

**Multiple types supported:**

**Selector:** A selector indicating which draggable elements are accepted.

**Function:** A function that will be called for each draggable on the page (passed as the first argument to the function). The function must return `true` if the draggable should be accepted.

**Code examples:**

Initialize the droppable with the accept option specified:

```
1 | $( ".selector" ).droppable({ accept: ".s
```

Get or set the accept option, after initialization:

```
1 | // getter
2 | var accept = $( ".selector" ).droppable(
3 |
4 | // setter
5 | $( ".selector" ).droppable( "option", "a
```

**activeClass**

Type: [String](#)

Default: `false`

If specified, the class will be added to the droppable while an acceptable draggable is being dragged.

### Code examples:

Initialize the droppable with the `activeClass` option specified:

```
1 | $( ".selector" ).droppable({ activeClass: "ui-state-h
```

Get or set the `activeClass` option, after initialization:

```
1 | // getter
2 | var activeClass = $( ".selector" ).droppable(
3 |
4 | // setter
5 | $( ".selector" ).droppable( "option", "activeClass"
```

## addClasses

Type: Boolean

Default: `true`

If set to `false`, will prevent the `ui-droppable` class from being added. This may be desired as a performance optimization when calling `.droppable()` init on hundreds of elements.

### Code examples:

Initialize the droppable with the `addClasses` option specified:

```
1 | $( ".selector" ).droppable({ addClasses: false
```

Get or set the `addClasses` option, after initialization:

```
1 // getter
2 var addClasses = $( ".selector" ).droppable(
3
4 // setter
5 $( ".selector" ).droppable( "option", "addClasses"
```

## disabled

Type: [Boolean](#)

Default: `false`

Disables the droppable if set to `true`.

### Code examples:

Initialize the droppable with the disabled option specified:

```
1 | $( ".selector" ).droppable({ disabled: true });
```

Get or set the disabled option, after initialization:

```
1 // getter
2 var disabled = $( ".selector" ).droppable( "option"
3
4 // setter
5 $( ".selector" ).droppable( "option", "disabled"
```

## greedy

Type: [Boolean](#)

Default: `false`

By default, when an element is dropped on nested droppables, each droppable will receive the element. However, by setting this option to `true`, any parent droppables will not receive the element.

### Code examples:

Initialize the droppable with the greedy option specified:

```
1 | $( ".selector" ).droppable({ greedy: true });
```

Get or set the greedy option, after initialization:

```
1 | // getter
2 | var greedy = $( ".selector" ).droppable( "option"
3 |
4 | // setter
5 | $( ".selector" ).droppable( "option", "greedy"
```

## hoverClass

Type: [String](#)

Default: `false`

If specified, the class will be added to the droppable while an acceptable draggable is being hovered over the droppable.

### Code examples:

Initialize the droppable with the hoverClass option specified:

```
1 | $( ".selector" ).droppable({ hoverClass: "drop-hover"
```

Get or set the hoverClass option, after initialization:

```
1 | // getter
2 | var hoverClass = $( ".selector" ).droppable(
3 |
4 | // setter
```

```
5 $( ".selector" ).droppable( "option", "hoverClass"
```

## scope

Type: [String](#)

Default: `"default"`

Used to group sets of draggable and droppable items, in addition to the `accept` option. A draggable with the same scope value as a droppable will be accepted.

### Code examples:

Initialize the droppable with the scope option specified:

```
1 $( ".selector" ).droppable({ scope: "tasks" });
```

Get or set the scope option, after initialization:

```
1 // getter
2 var scope = $( ".selector" ).droppable( "option"
3
4 // setter
5 $( ".selector" ).droppable( "option", "scope"
```

## tolerance

Type: [String](#)

Default: `"intersect"`

Specifies which mode to use for testing whether a draggable is hovering over a droppable. Possible values:

`"fit"`: Draggable overlaps the droppable entirely.

`"intersect"`: Draggable overlaps the droppable at least 50% in both directions.

"pointer": Mouse pointer overlaps the droppable.

"touch": Draggable overlaps the droppable any amount.

### Code examples:

Initialize the droppable with the tolerance option specified:

```
1 | $( ".selector" ).droppable({ tolerance: "fit"
```

Get or set the tolerance option, after initialization:

```
1 | // getter  
2 | var tolerance = $( ".selector" ).droppable( "option"  
3 |  
4 | // setter  
5 | $( ".selector" ).droppable( "option", "tolerance"
```

## Methods

### destroy()

Removes the droppable functionality completely. This will return the element back to its pre-init state.

This method does not accept any arguments.

#### Code examples:

Invoke the destroy method:

```
1 | $( ".selector" ).droppable( "destroy" );
```

### disable()

Disables the droppable.

This method does not accept any arguments.

#### Code examples:

Invoke the disable method:

```
1 | $( ".selector" ).droppable( "disable" );
```

### enable()

Enables the droppable.

This method does not accept any arguments.

### Code examples:

Invoke the enable method:

```
1 | $( ".selector" ).droppable( "enable" );
```

## option( optionName )

Returns: *Object*

Gets the value currently associated with the specified `optionName`.

**optionName**

Type: *String*

The name of the option to get.

### Code examples:

Invoke the method:

```
1 | var isDisabled = $( ".selector" ).droppable(
```

## option()

Returns: *PlainObject*

Gets an object containing key/value pairs representing the current droppable options hash.

This method does not accept any arguments.

### Code examples:

Invoke the method:

```
1 | var options = $( ".selector" ).droppable( "option"
```

## option( optionName, value )

Sets the value of the droppable option associated with the specified `optionName`.

### **optionName**

Type: String

The name of the option to set.

### **value**

Type: Object

A value to set for the option.

### **Code examples:**

Invoke the method:

```
1 | $( ".selector" ).droppable( "option", "disabled"
```

## option( options )

Sets one or more options for the droppable.

### **options**

Type: Object

A map of option-value pairs to set.

### **Code examples:**

Invoke the method:

```
1 | $( ".selector" ).droppable( "option", { disabled:
```

## widget()

Returns: *jQuery*

Returns a `jQuery` object containing the droppable element.  
This method does not accept any arguments.

### Code examples:

Invoke the `widget` method:

```
1 | var widget = $( ".selector" ).droppable( "widget"
```

## Events

**activate( event, ui )**

Type: `dropactivate`

Triggered when an accepted draggable starts dragging. This can be useful if you want to make the droppable "light up" when it can be dropped on.

**event**

Type: `Event`

**ui**

Type: `Object`

**draggable**

Type: `jQuery`

A jQuery object representing the draggable element.

**helper**

Type: `jQuery`

A jQuery object representing the helper that is being dragged.

**position**

Type: `Object`

Current CSS position of the draggable helper as `{ top, left }` object.

**offset**

Type: `Object`

Current offset position of the draggable helper as `{ top, left }` object.

### Code examples:

Initialize the droppable with the activate callback specified:

```
1 | $( ".selector" ).droppable({
2 |   activate: function( event, ui ) {}
3 | });
```

Bind an event listener to the dropactivate event:

```
1 | $( ".selector" ).on( "dropactivate", function
```

**create( event, ui )** *Type:* `dropcreate`

Triggered when the droppable is created.

**event**  
Type: [Event](#)

**ui**  
Type: [Object](#)

### Code examples:

Initialize the droppable with the create callback specified:

```
1 | $( ".selector" ).droppable({
2 |   create: function( event, ui ) {}
3 | });
```

Bind an event listener to the dropcreate event:

```
1 | $( ".selector" ).on( "dropcreate", function( event, ui
```

## deactivate( event, ui )

Type: `dropdeactivate`

Triggered when an accepted draggable stops dragging.

### event

Type: `Event`

### ui

Type: `Object`

#### draggable

Type: `jQuery`

A jQuery object representing the draggable element.

#### helper

Type: `jQuery`

A jQuery object representing the helper that is being dragged.

#### position

Type: `Object`

Current CSS position of the draggable helper as `{ top, left }` object.

#### offset

Type: `Object`

Current offset position of the draggable helper as `{ top, left }` object.

### Code examples:

Initialize the droppable with the deactivate callback specified:

```
1 | $( ".selector" ).droppable({
2 |   deactivate: function( event, ui ) {}
3 | });
```

Bind an event listener to the dropdeactivate event:

```
1 | $( ".selector" ).on( "dropdeactivate", function
```

## drop( event, ui )

Type: `drop`

Triggered when an accepted draggable is dropped on the droppable (based on the `tolerance` option).

### event

Type: `Event`

### ui

Type: `Object`

#### draggable

Type: `jQuery`

A jQuery object representing the draggable element.

#### helper

Type: `jQuery`

A jQuery object representing the helper that is being dragged.

#### position

Type: `Object`

Current CSS position of the draggable helper as `{ top, left }` object.

#### offset

Type: `Object`

Current offset position of the draggable helper as `{ top, left }` object.

**Code examples:**

Initialize the droppable with the drop callback specified:

```
1 | $( ".selector" ).droppable({
2 |   drop: function( event, ui ) {}
3 | });
```

Bind an event listener to the drop event:

```
1 | $( ".selector" ).on( "drop", function( event, ui ) {
```

**out( event, ui )**

Type: `dropout`

Triggered when an accepted draggable is dragged out of the droppable (based on the `tolerance` option).

**event**

Type: `Event`

**ui**

Type: `Object`

**Code examples:**

Initialize the droppable with the out callback specified:

```
1 | $( ".selector" ).droppable({
2 |   out: function( event, ui ) {}
3 | });
```

Bind an event listener to the dropout event:

```
1 | $( ".selector" ).on( "dropout", function( event, ui )
```

## over( event, ui )

Type: `dropover`

Triggered when an accepted draggable is dragged over the droppable (based on the `tolerance` option).

### event

Type: `Event`

### ui

Type: `Object`

#### draggable

Type: `jQuery`

A jQuery object representing the draggable element.

#### helper

Type: `jQuery`

A jQuery object representing the helper that is being dragged.

#### position

Type: `Object`

Current CSS position of the draggable helper as `{ top, left }` object.

#### offset

Type: `Object`

Current offset position of the draggable helper as `{ top, left }` object.

### Code examples:

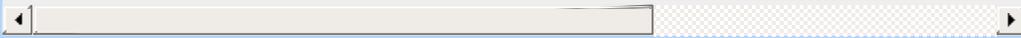
Initialize the droppable with the over callback specified:

```
1 | $( ".selector" ).droppable({  
2 |   over: function( event, ui ) {}
```

```
3 | });
```

Bind an event listener to the droptop event:

```
1 | $( ".selector" ).on( "droptop", function( event, ui
```



## Example:

*A pair of draggable and droppable elements.*

```
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>droppable demo</title>
6   <link rel="stylesheet" href="http://code.jqu
7   <style>
8     #draggable {
9       width: 100px;
10      height: 100px;
11      background: #ccc;
12    }
13    #droppable {
14      position: absolute;
15      left: 250px;
16      top: 0;
17      width: 125px;
18      height: 125px;
19      background: #999;
20      color: #fff;
21      padding: 10px;
22    }
23  </style>
24  <script src="http://code.jquery.com/jquery-:
25  <script src="http://code.jquery.com/ui/1.10
26 </head>
27 <body>
28
29 <div id="droppable">Drop here</div>
30 <div id="draggable">Drag me</div>
31
```

```
32 <script>
33 $( "#draggable" ).draggable();
34 $( "#droppable" ).droppable({
35   drop: function() {
36     alert( "dropped" );
37   }
38 });
39 </script>
40
41 </body>
42 </html>
```

Demo

POWERED BYHERONOTE

A new version of this book is available!

---

# .effect()

Categories: [Effects](#) | [Effects Core](#) | [Methods](#)

`.effect( effect [, options ] [, duration ] [, complete ] )`

Returns: `jQuery`

**Description:** *Apply an animation effect to an element.*

**`.effect( effect [, options ] [, duration ] [, complete ] )`**

**effect**

Type: `String`

A string indicating which `effect` to use for the transition.

**options**

Type: `Object`

Effect-specific settings and `easing`.

**duration** (default: `400`)

Type: `Number` or `String`

A string or number determining how long the animation will run.

**complete**

Type: `Function()`

A function to call once the animation is complete.

**`.effect( options )`**

**options**

Type: `Object`

All animation settings. The only required property is `effect`.

**effect**

Type: `String`

A string indicating which `effect` to use for the transition.

**easing** (default: `"swing"`)

Type: `String`

A string indicating which `easing` function to use for the transition.

**duration** (default: `400`)

Type: `Number` or `String`

A string or number determining how long the animation will

run.

---

**complete**

Type: Function()

A function to call once the animation is complete.

The `.effect()` method applies a named animation effect to an element. Many effects also support a show or hide mode, which can be accomplished with the `.show()`, `.hide()`, and `.toggle()` methods.

## Example:

*Apply the bounce effect to a div.*

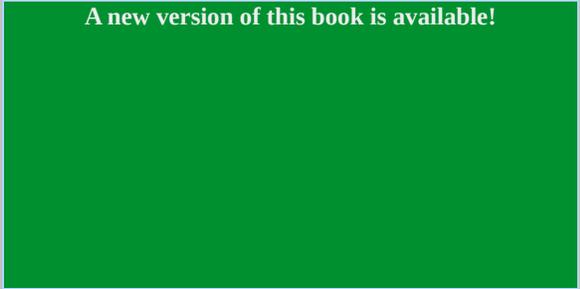
```
1  <!doctype html>
2  <html lang="en">
3  <head>
4    <meta charset="utf-8">
5    <title>effect demo</title>
6    <link rel="stylesheet" href="http://code.jq
7    <style>
8    div {
9      width: 100px;
10     height: 100px;
11     background: #ccc;
12     border: 1px solid #000;
13   }
14   </style>
15   <script src="http://code.jquery.com/jquery-:
16   <script src="http://code.jquery.com/ui/1.10
17 </head>
18 <body>
19
20 <p>Click anywhere to apply the effect.</p>
21 <div></div>
22
23 <script>
24 $( document ).click(function() {
25   $( "div" ).effect( "bounce", "slow" );
26 });
27 </script>
28
29 </body>
30 </html>
```



Demo

**POWERED BYHERONOTE**

A new version of this book is available!



---

# .enableSelection()

Categories: [Methods](#) | [UI Core](#)

`.enableSelection()`

Returns: `jQuery`

**Description:** *Enable selection of text content within the set of matched elements.*

**`.enableSelection()`**

version added: 1.6, deprecated: 1.9

This method does not accept any arguments.

The `.enableSelection()` method can be used to re-enable selection of text that was disabled via `.disableSelection()`.

POWERED BYHERONOTE

A new version of this book is available!

---

# Explode Effect

Categories: [Effects](#)

## Explode Effect

**Description:** *The explode effect hides or shows an element by splitting it into pieces.*

### explode

**pieces** (default: 9)

Type: Integer

The number of pieces to explode, should be a perfect square, any other values are rounded to the nearest square.

## Example:

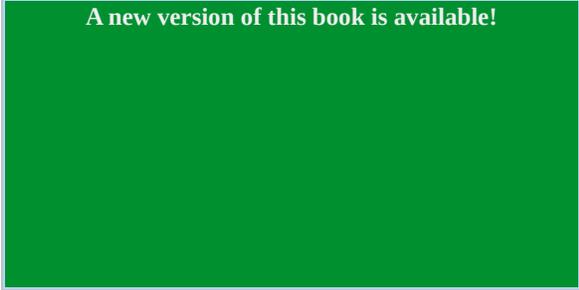
*Toggle a div using the explode effect.*

```
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>explode demo</title>
6   <link rel="stylesheet" href="http://code.jq
7   <style>
8     #toggle {
9       width: 100px;
10      height: 100px;
11      background: #ccc;
12    }
13  </style>
14  <script src="http://code.jquery.com/jquery-:
15  <script src="http://code.jquery.com/ui/1.10
16 </head>
17 <body>
18
19 <p>Click anywhere to toggle the box.</p>
20 <div id="toggle"></div>
21
22 <script>
23 $( document ).click(function() {
24   $( "#toggle" ).toggle( "explode" );
25 });
26 </script>
27
28 </body>
29 </html>
```

Demo

POWERED BYHERONOTE

A new version of this book is available!



---

# Fade Effect

Categories: [Effects](#)

---

## Fade Effect

**Description:** *The fade effect hides or shows an element by fading it.*

**fade**

## Example:

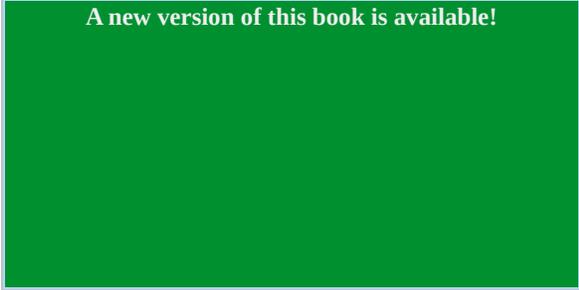
*Toggle a div using the fade effect.*

```
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>fade demo</title>
6   <link rel="stylesheet" href="http://code.jq
7   <style>
8   #toggle {
9     width: 100px;
10    height: 100px;
11    background: #ccc;
12  }
13 </style>
14 <script src="http://code.jquery.com/jquery-:
15 <script src="http://code.jquery.com/ui/1.10
16 </head>
17 <body>
18
19 <p>Click anywhere to toggle the box.</p>
20 <div id="toggle"></div>
21
22 <script>
23 $( document ).click(function() {
24   $( "#toggle" ).toggle( "fade" );
25 });
26 </script>
27
28 </body>
29 </html>
```

Demo

POWERED BYHERONOTE

A new version of this book is available!



---

# .focus()

Categories: [Method Overrides](#) | [Methods](#) | [UI Core](#)

`.focus( delay [, callback ] )`

Returns: `jQuery`

**Description:** *Asynchronously set focus to an element.*

**`.focus( delay [, callback ] )`**

**delay**

Type: `Integer`

The number of milliseconds to wait before setting focus.

**callback**

Type: `Function()`

A function to invoke after the element has been focused.

This plugin extends jQuery's built-in `.focus()` method. If jQuery UI is not loaded, calling the `.focus()` method may not fail directly, as the method still exists. However, the expected behavior will not occur.

POWERED BYHERONOTE

A new version of this book is available!

---

# :focusable Selector

Categories: [Selectors](#) | [UI Core](#)

## focusable selector

**Description:** *Selects elements which can be focused.*

### jQuery( ":focusable" )

Some elements are natively focusable, while others require explicitly setting a tab index. In all cases, the element must be visible in order to be focusable.

Elements of the following type are focusable if they are not disabled: `input`, `select`, `textarea`, `button`, and `object`. Anchors are focusable if they have an `href` or `tabindex` attribute. `area` elements are focusable if they are inside a named map, have an `href` attribute, and there is a visible image using the map. All other elements are focusable based solely on their `tabindex` attribute and visibility.

*Note: Elements with a negative tab index are `:focusable`, but not `:tabbable`.*

## Example:

*Select focusable elements and highlight them with a red border.*

```
1  <!doctype html>
2  <html lang="en">
3  <head>
4      <meta charset="utf-8">
5      <title>focusable demo</title>
6      <link rel="stylesheet" href="http://code.jq
7      <style>
8      input, a, p {
9          border: 1px solid #000;
10     }
11     div {
12         padding: 5px;
13     }
14 </style>
15 <script src="http://code.jquery.com/jquery-:
16 <script src="http://code.jquery.com/ui/1.10
17 </head>
18 <body>
19
20 <div><input value="text input"></div>
21 <div><a>anchor without href</a></div>
22 <div><a href="#">anchor with href</a></div>
23 <div><p>paragraph without tabindex</p></div>
24 <div><p tabindex="1">paragraph with tabindex<
25
26 <script>
27 $( ":focusable" ).css( "border-color", "red"
28 </script>
29
30 </body>
```

31 | `</html>`

Demo

POWERED BYHERONOTE

A new version of this book is available!

---

# Fold Effect

Categories: [Effects](#)

## Fold Effect

**Description:** *The fold effect hides or shows an element by folding it.*

### fold

**size** (default: `15`)

Type: Number or String

The size of the "folded" element.

**horizFirst** (default: `false`)

Type: Boolean

Whether the horizontal direction happens first when hiding. Remember, showing inverts hiding.

## Example:

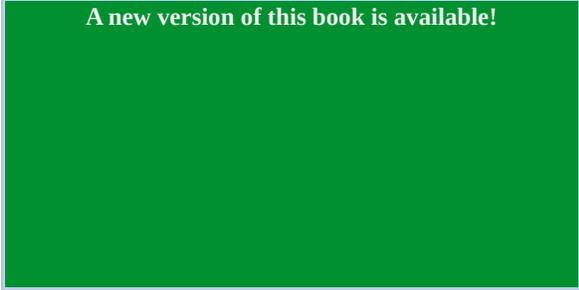
*Toggle a div using the fold effect.*

```
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>fold demo</title>
6   <link rel="stylesheet" href="http://code.jq
7   <style>
8     #toggle {
9       width: 100px;
10      height: 100px;
11      background: #ccc;
12    }
13  </style>
14  <script src="http://code.jquery.com/jquery-:
15  <script src="http://code.jquery.com/ui/1.10
16 </head>
17 <body>
18
19 <p>Click anywhere to toggle the box.</p>
20 <div id="toggle"></div>
21
22 <script>
23 $( document ).click(function() {
24   $( "#toggle" ).toggle( "fold" );
25 });
26 </script>
27
28 </body>
29 </html>
```

Demo

POWERED BYHERONOTE

A new version of this book is available!



---

# .hide()

Categories: [Effects](#) | [Effects Core](#) | [Method Overrides](#) | [Methods](#)

```
.hide( effect [, options ] [, duration ] [, complete ] )
```

Returns: `jQuery`

**Description:** *Hide the matched elements, using custom effects.*

```
.hide( effect [, options ] [, duration ] [, complete ] )
```

**effect**

Type: `String`

A string indicating which `effect` to use for the transition.

**options**

Type: `Object`

Effect-specific settings and `easing`.

**duration** (default: `400`)

Type: `Number` or `String`

A string or number determining how long the animation will run.

**complete**

Type: `Function()`

A function to call once the animation is complete.

```
.hide( options )
```

**options**

Type: `Object`

All animation settings. The only required property is `effect`.

**effect**

Type: `String`

A string indicating which `effect` to use for the transition.

**easing** (default: `"swing"`)

Type: `String`

A string indicating which `easing` function to use for the transition.

**duration** (default: `400`)

Type: `Number` or `String`

A string or number determining how long the animation will run.

**complete**

Type: `Function()`

A function to call once the animation is complete.

This plugin extends jQuery's built-in `.hide()` method. If jQuery UI is not loaded, calling the `.hide()` method may not fail directly, as the method still exists. However, the expected behavior will not occur.

## Example:

*Hide a div using the drop effect.*

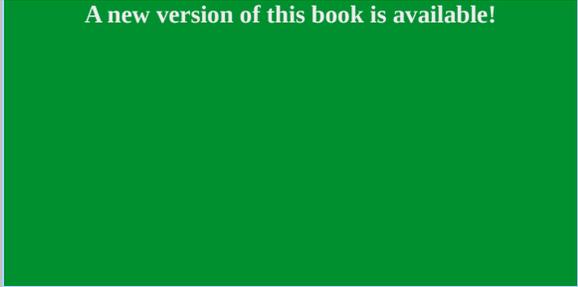
```
1  <!doctype html>
2  <html lang="en">
3  <head>
4      <meta charset="utf-8">
5      <title>hide demo</title>
6      <link rel="stylesheet" href="http://code.jq
7      <style>
8      div {
9          width: 100px;
10         height: 100px;
11         background: #ccc;
12         border: 1px solid #000;
13     }
14     </style>
15     <script src="http://code.jquery.com/jquery-:
16     <script src="http://code.jquery.com/ui/1.10
17 </head>
18 <body>
19
20 <button>hide the div</button>
21 <div></div>
22
23 <script>
24 $( "button" ).click(function() {
25     $( "div" ).hide( "drop", { direction: "down
26 });
27 </script>
28
29 </body>
30 </html>
```



Demo

POWERED BYHERONOTE

A new version of this book is available!



---

# Highlight Effect

Categories: [Effects](#)

## Highlight Effect

**Description:** *The highlight effect hides or shows an element by animating its background color first.*

### highlight

**color** (default: "#ffff99")

Type: String

The background color used during the animation.

## Example:

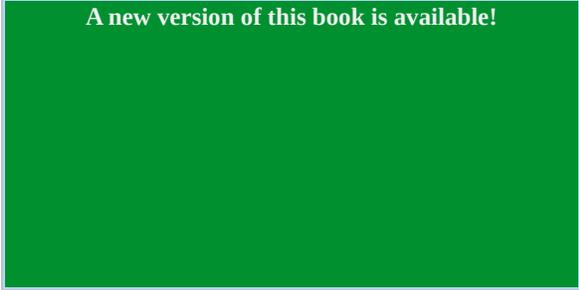
*Toggle a div using the highlight effect.*

```
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>highlight demo</title>
6   <link rel="stylesheet" href="http://code.jq
7   <style>
8     #toggle {
9       width: 100px;
10      height: 100px;
11      background: #ccc;
12    }
13  </style>
14  <script src="http://code.jquery.com/jquery-:
15  <script src="http://code.jquery.com/ui/1.10
16 </head>
17 <body>
18
19 <p>Click anywhere to toggle the box.</p>
20 <div id="toggle"></div>
21
22 <script>
23 $( document ).click(function() {
24   $( "#toggle" ).toggle( "highlight" );
25 });
26 </script>
27
28 </body>
29 </html>
```

Demo

POWERED BYHERONOTE

A new version of this book is available!



---

# Widget Factory

Categories: [Utilities](#) | [Widgets](#)

## Contents:

[jQuery.widget\( name \[, base \], prototype \)](#)

[jQuery.widget\( name \[, base \], prototype \)](#)

[jQuery.Widget](#)

```
jQuery.widget( name [, base ], prototype )
```

**Description:** *Create stateful jQuery plugins using the same abstraction as all jQuery UI widgets.*

```
jQuery.widget( name [, base ], prototype )
```

**name**

Type: [String](#)

The name of the widget to create, including the namespace.

**base**

Type: [Function\(\)](#)

The base widget to inherit from. This must be a constructor that can be instantiated with the `new` keyword. Defaults to

`jQuery.Widget`.

**prototype**

Type: [PlainObject](#)

The object to use as a prototype for the widget.

You can create new widgets from scratch, using just the `$.Widget` object as a base to inherit from, or you can explicitly inherit from existing jQuery UI or third-party widgets. Defining a widget with the same name as you inherit from even allows you to extend widgets in place.

jQuery UI contains many widgets that maintain state and therefore have a slightly different usage pattern than typical jQuery plugins. All of jQuery UI's widgets use the same patterns, which is defined by the widget factory. So if you learn how to use one widget, then you'll know how to use all of them.

*Note: This documentation shows examples using the [progressbar widget](#) but the syntax is the same for every widget.*

## Initialization

In order to track the state of the widget, we must introduce a full life cycle for the widget. The life cycle starts when the widget is

initialized. To initialize a widget, we simply call the plugin on one or more elements.

```
1 | $( "#elem" ).progressbar();
```

This will initialize each element in the jQuery object, in this case the element with an id of `"elem"`. Because we called the `progressbar()` method with no parameters, the widget is initialized with its default options. We can pass a set of options during initialization in order to override the default options.

```
1 | $( "#elem" ).progressbar({ value: 20 });
```

We can pass as many or as few options as we want during initialization. Any options that we don't pass will just use their default values.

The options are part of the widget's state, so we can set options after initialization as well. We'll see this later with the `option` method.

## Methods

Now that the widget is initialized, we can query its state or perform actions on the widget. All actions after initialization take the form of a method call. To call a method on a widget, we pass the name of the method to the jQuery plugin. For example, to call the `value()` method on our progressbar widget, we would use:

```
1 | $( "#elem" ).progressbar( "value" );
```

If the method accepts parameters, we can pass them after the method name. For example, to pass the parameter `40` to the `value()` method, we can use:

```
1 | $( "#elem" ).progressbar( "value", 40 );
```

Just like other methods in jQuery, most widget methods return the jQuery object for chaining.

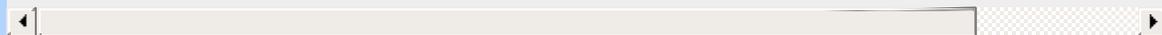
```
1 | $( "#elem" )
2 |   .progressbar( "value", 90 )
3 |   .addClass( "almost-done" );
```

Each widget will have its own set of methods based on the functionality that the widget provides. However, there are a few methods that exist on all widgets, which are documented below.

## Events

All widgets have events associated with their various behaviors to notify you when the state is changing. For most widgets, when the events are triggered, the names are prefixed with the widget name. For example, we can bind to progressbar's `change` event which is triggered whenever the value changes.

```
1 | $( "#elem" ).bind( "progressbarchange", funct:
2 |   alert( "The value has changed!" );
3 | });
```



Each event has a corresponding callback, which is exposed as an option. We can hook into progressbar's `change` callback instead of binding to the `progressbarchange` event, if we want to.

```
1 | $( "#elem" ).progressbar( {
2 |   change: function() {
3 |     alert( "The value has changed!" );
4 |   }
5 | });
```

All widgets have a `create` event which is triggered upon instantiation.

---

## Base Widget

**Description:** *The base widget used by the widget factory.*

# QuickNav

## Options

[disabled](#)  
[hide](#)  
[show](#)

## Methods

[destroy](#)  
[disable](#)  
[enable](#)  
[option](#)  
[widget](#)  
[\\_\\_create](#)  
[\\_\\_destroy](#)  
[\\_\\_getCreateEventData](#)  
[\\_\\_getCreateOptions](#)  
[\\_\\_init](#)  
[\\_\\_setOptions](#)  
[\\_\\_setOption](#)  
[\\_\\_on](#)  
[\\_\\_off](#)  
[\\_\\_super](#)  
[\\_\\_superApply](#)  
[\\_\\_delay](#)  
[\\_\\_hoverable](#)  
[\\_\\_focusable](#)  
[\\_\\_trigger](#)  
[\\_\\_show](#)  
[\\_\\_hide](#)

## Events

[create](#)

# Options

## disabled

Type: [Boolean](#)

Default: `false`

Disables the widget if set to `true`.

### Code examples:

Initialize the widget with the disabled option specified:

```
1 | $( ".selector" ).widget({ disabled: true
```

Get or set the disabled option, after initialization:

```
1 | // getter
2 | var disabled = $( ".selector" ).widget(
3 |
4 | // setter
5 | $( ".selector" ).widget( "option", "disa
```

## hide Type: [Boolean](#) or [Number](#) or [String](#) or [Object](#)

Default: `null`

If and how to animate the hiding of the element.

### Multiple types supported:

**Boolean:** When set to `false`, no animation will be used and the element will be hidden immediately. When set to `true`, the element will fade out with the default duration and the default easing.

**Number:** The element will fade out with the specified duration and the default easing.

**String:** The element will be hidden using the specified effect. The value can either be the name of a built-in jQuery animation method, such as `"slideUp"`, or the name of a jQuery UI effect, such as `"fold"`. In either case the effect will be used with the default duration and the default easing.

**Object:** If the value is an object, then `effect`, `delay`, `duration`, and `easing` properties may be provided. If the `effect` property contains the name of a jQuery method, then that method will be used; otherwise it is assumed to be the name of a jQuery UI effect. When using a jQuery UI effect that supports additional settings, you may include those settings in the object and they will be passed to the effect. If `duration` or `easing` is omitted, then the default values will be used. If `effect` is omitted, then `"fadeOut"` will be used. If `delay` is omitted, then no delay is used.

### Code examples:

Initialize the widget with the hide option specified:

```
1 | $( ".selector" ).widget({ hide: { effect: "explode" }
```

Get or set the hide option, after initialization:

```
1 | // getter
2 | var hide = $( ".selector" ).widget( "option",
3 |
4 | // setter
5 | $( ".selector" ).widget( "option", "hide", { effect:
```

**show** Type: [Boolean](#) or [Number](#) or [String](#) or [Object](#)

Default: `null`



```
5 $( ".selector" ).widget( "option", "show", { effect:
```



## Methods

### `_create()`

The `_create()` method is the widget's constructor. There are no parameters, but `this.element` and `this.options` are already set.

This method does not accept any arguments.

### `_delay( fn [, delay ] )`

Returns: *Number*

Invokes the provided function after a specified delay. Keeps `this` context correct. Essentially `setTimeout()`.

Returns the timeout ID for use with `clearTimeout()`.

#### **fn**

Type: *Function()* or *String*

The function to invoke. Can also be the name of a method on the widget.

#### **delay**

Type: *Number*

The number of milliseconds to wait before invoking the function. Defaults to `0`.

### `_destroy()`

The public `destroy()` method cleans up all common data, events, etc. and then delegates out to `_destroy()` for custom, widget-specific, cleanup.

This method does not accept any arguments.

## **\_focusable( element )**

Sets up `element` to apply the `ui-state-focus` class on focus. The event handlers are automatically cleaned up on destroy.

### **element**

Type: [jQuery](#)

The element(s) to apply the focusable behavior to.

## **\_getCreateEventData()** *Returns: [Object](#)*

All widgets trigger the `create` event. By default, no data is provided in the event, but this method can return an object which will be passed as the `create` event's data.

This method does not accept any arguments.

## **\_getCreateOptions()** *Returns: [Object](#)*

This method allows the widget to define a custom method for defining options during instantiation. This user-provided options override the options returned by this method which override the default options.

This method does not accept any arguments.

## **\_hide( element, option [, callback ] )**

Hides an element immediately, using built-in animation methods, or using custom effects. See the [hide](#) option for possible `option` values.

### **element**

Type: [jQuery](#)

The element(s) to hide.

**option**Type: [Object](#)

The settings defining how to hide the element.

**callback**Type: [Function\(\)](#)

Callback to invoke after the element has been fully hidden.

---

## **`_hoverable( element )`**

Sets up `element` to apply the `ui-state-hover` class on hover. The event handlers are automatically cleaned up on destroy.

**element**Type: [jQuery](#)

The element(s) to apply the hoverable behavior to.

---

## **`_init()`**

Widgets have the concept of initialization that is distinct from creation. Any time the plugin is called with no arguments or with only an option hash, the widget is initialized; this includes when the widget is created.

*Note: Initialization should only be handled if there is a logical action to perform on successive calls to the widget with no arguments.*

This method does not accept any arguments.

---

## **`_off( element, eventName )`**

Unbinds event handlers from the specified element(s).

**element**Type: [jQuery](#)

The element(s) to unbind the event handlers from.

Unlike the `_on()` method, the elements are required for `_off()`.**eventName**Type: [String](#)

One or more space-separated event types.

---

## `_on( [element ], handlers )`

Binds event handlers to the specified element(s). Delegation is supported via selectors inside the event names, e.g., "`click .foo`". The `_on()` method provides several benefits of direct event binding:

- Maintains proper `this` context inside the handlers.

- Automatically handles disabled widgets: If the widget is disabled or the event occurs on an element with the `ui-state-disabled` class, the event handler is not invoked.

- Event handlers are automatically namespaced and cleaned up on destroy.

**element**Type: [jQuery](#)Which element(s) to bind the event handlers to. If no element is provided, `this.element` is used.**handlers**Type: [Object](#)

A map in which the string keys represent the event type and optional selector for delegation, and the values represent a handler function to be called for the event.

## `_setOption( key, value )`

Called from the `setOptions()` method for each individual option. Widget state should be updated based on changes.

**key**

Type: `String`

The name of the option to set.

**value**

Type: `Object`

A value to set for the option.

---

## `_setOptions( options )`

Called whenever the `option()` method is called, regardless of the form in which the `option()` method was called. Overriding this is useful if you can defer processor-intensive changes for multiple option changes.

**options**

Type: `Object`

A map of option-value pairs to set.

---

## `_show( element, option [, callback ] )`

Shows an element immediately, using built-in animation methods, or using custom effects. See the `show` option for possible `option` values.

**element**

Type: `jQuery`

The element(s) to show.

**option**

Type: `Object`

The settings defining how to show the element.

**callback**

Type: [Function\(\)](#)

Callback to invoke after the element has been fully shown.

---

**\_super()**

Invokes the method of the same name from the parent widget, with any specified arguments. Essentially `.call()`.

This method does not accept any arguments.

---

**\_superApply( arguments )**

Invokes the method of the same name from the parent widget, with the array of arguments. Essentially `.apply()`.

**arguments**

Type: [Array](#)

Array of arguments to pass to the parent method.

---

**\_trigger( type [, event ] [, data ] )**

Triggers an event and its associated callback.

The option with the name equal to type is invoked as the callback.

The event name is the widget name + type.

*Note: When providing data, you must provide all three parameters. If there is no event to pass along, just pass `null`.*

**type**

Type: [String](#)

The `type` should match the name of a callback option. The full event type will be generated automatically.

### **event**

Type: Event

The original event that caused this event to occur; useful for providing context to the listener.

### **data**

Type: Object

A hash of data associated with the event.

---

## **destroy()**

Removes the widget functionality completely. This will return the element back to its pre-init state.

This method does not accept any arguments.

---

## **disable()**

Disables the widget.

This method does not accept any arguments.

---

## **enable()**

Enables the widget.

This method does not accept any arguments.

---

## **option( optionName )**

Returns: Object

Gets the value currently associated with the specified

`optionName`.

### **optionName**

Type: [String](#)

The name of the option to get.

### **Code examples:**

Invoke the method:

```
1 | var isDisabled = $( ".selector" ).widget( "option"
```

## **option()**

Returns: [PlainObject](#)

Gets an object containing key/value pairs representing the current widget options hash.

This method does not accept any arguments.

### **Code examples:**

Invoke the method:

```
1 | var options = $( ".selector" ).widget( "option"
```

## **option( optionName, value )**

Sets the value of the widget option associated with the specified `optionName`.

### **optionName**

Type: [String](#)

The name of the option to set.

### **value**

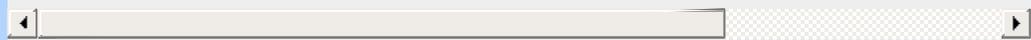
Type: [Object](#)

A value to set for the option.

### Code examples:

Invoke the method:

```
1 | $( ".selector" ).widget( "option", "disabled"
```



## option( options )

Sets one or more options for the widget.

### options

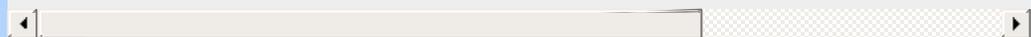
Type: Object

A map of option-value pairs to set.

### Code examples:

Invoke the method:

```
1 | $( ".selector" ).widget( "option", { disabled:
```



## widget()

Returns: *jQuery*

Returns a `jQuery` object containing the original element or other relevant generated element.

This method does not accept any arguments.

## Events

**create( event, ui )**

Type: `widgetcreate`

Triggered when the widget is created.

**event**

Type: `Event`

**ui**

Type: `Object`

### Code examples:

Initialize the widget with the create callback specified:

```
1 | $( ".selector" ).widget({  
2 |   create: function( event, ui ) {}  
3 | });
```

Bind an event listener to the widgetcreate event:

```
1 | $( ".selector" ).on( "widgetcreate", fu
```

**A new version of this book is available!**

---

# Menu Widget

Categories: [Widgets](#)

---

Menu Widget

version added: 1.9

**Description:** *Themeable menu with mouse and keyboard interactions for navigation.*

# QuickNav

## Examples

### Options

[disabled](#)  
[icons](#)  
[menus](#)  
[position](#)  
[role](#)

### Methods

[blur](#)  
[collapse](#)  
[collapseAll](#)  
[destroy](#)  
[disable](#)  
[enable](#)  
[expand](#)  
[focus](#)  
[isFirstItem](#)  
[isLastItem](#)  
[next](#)  
[nextPage](#)  
[option](#)  
[previous](#)  
[previousPage](#)  
[refresh](#)  
[select](#)  
[widget](#)

### Events

[blur](#)  
[create](#)  
[focus](#)  
[select](#)

A menu can be created from any valid markup as long as the elements have a strict parent/child relationship and each menu item has an anchor. The most commonly used element is the unordered list (`<ul>`):

```
<ul id="menu">
  <li><a href="#">Item 1</a></li>
  <li><a href="#">Item 2</a></li>
  <li><a href="#">Item 3</a>
    <ul>
      <li><a href="#">Item 3-1</a></li>
      <li><a href="#">Item 3-2</a></li>
      <li><a href="#">Item 3-3</a></li>
      <li><a href="#">Item 3-4</a></li>
      <li><a href="#">Item 3-5</a></li>
    </ul>
  </li>
  <li><a href="#">Item 4</a></li>
</ul>
```

```
<li><a href="#">Item 5</a></li>
</ul>
```

If you use a structure other than `<ul>/<li>`, including using the same element for the menu and the menu items, use the `menus` option to specify a way to differentiate the two elements, e.g., `menus: "div.menuElement"`.

Any menu item can be disabled by adding the `ui-state-disabled` class to that element.

To add icons to the menu, include them in the markup:

```
<ul id="menu">
  <li><a href="#"><span class="ui-icon ui-icon-disk">
</span>Save</a></li>
</ul>
```

Menu automatically adds the necessary padding to items without icons.

## Keyboard interaction

**ENTER/SPACE:** Invoke the focused menu item's action, which may be opening a submenu.

**UP:** Move focus to the previous menu item.

**DOWN:** Move focus to the next menu item.

**RIGHT:** Open the submenu, if available.

**LEFT:** Close the current submenu and move focus to the parent menu item. If not in a submenu, do nothing.

**ESCAPE:** Close the current submenu and move focus to the parent menu item. If not in a submenu, do nothing.

Typing a letter moves focus to the first item whose title starts with that character. Repeating the same character cycles through matching items. Typing more characters within the one second timer matches those characters.

Disabled items can receive keyboard focus, but do not allow any other interaction.

## Dependencies

[UI Core](#)

[Widget Factory](#)

[Position](#)

## **Additional Notes:**

This widget requires some functional CSS, otherwise it won't work. If you build a custom theme, use the widget's specific CSS file as a starting point.

## Options

### disabled

Type: [Boolean](#)

Default: `false`

Disables the menu if set to `true`.

#### Code examples:

Initialize the menu with the disabled option specified:

```
1 | $( ".selector" ).menu({ disabled: true
```

Get or set the disabled option, after initialization:

```
1 | // getter
2 | var disabled = $( ".selector" ).menu( "o
3 |
4 | // setter
5 | $( ".selector" ).menu( "option", "disabi
```

### icons

Type: [Object](#)

Default: `{ submenu: "ui-icon-carat-1-e" }`

Icons to use for submenus, matching an icon defined by the jQuery UI CSS Framework.

submenu (string, default: "ui-icon-carat-1-e")

#### Code examples:

Initialize the menu with the icons option specified:

```
1 |
```

```
1 | $( ".selector" ).menu({ icons: { submenu: "ui-icon-c
```

Get or set the icons option, after initialization:

```
1 | // getter
2 | var icons = $( ".selector" ).menu( "option",
3 |
4 | // setter
5 | $( ".selector" ).menu( "option", "icons", { submenu:
```

**menus**

**Type:** [String](#)

**Default:** `"ul"`

Selector for the elements that serve as the menu container, including sub-menus.

*Note: The `menus` option should not be changed after initialization. Existing submenus will not be updated.*

### Code examples:

Initialize the menu with the menus option specified:

```
1 | $( ".selector" ).menu({ menus: "div" });
```

Get or set the menus option, after initialization:

```
1 | // getter
2 | var menus = $( ".selector" ).menu( "option",
3 |
4 | // setter
5 | $( ".selector" ).menu( "option", "menus", "div"
```

## position

Type: [Object](#)

Default: `{ my: "left top", at: "right top" }`

Identifies the position of submenus in relation to the associated parent menu item. The `of` option defaults to the parent menu item, but you can specify another element to position against. You can refer to the [jQuery UI Position](#) utility for more details about the various options.

### Code examples:

Initialize the menu with the position option specified:

```
1 | $( ".selector" ).menu({ position: { my: "left top"
```

Get or set the position option, after initialization:

```
1 | // getter
2 | var position = $( ".selector" ).menu( "option"
3 |
4 | // setter
5 | $( ".selector" ).menu( "option", "position", { my:
```

## role

Type: [String](#)

Default: `"menu"`

Customize the ARIA roles used for the menu and menu items. The default uses `"menuitem"` for items. Setting the `role` option to `"listbox"` will use `"option"` for items. If set to `null`, no roles will be set, which is useful if the menu is being controlled by another element that is maintaining focus.

Note: The `role` option should not be changed after initialization. Existing (sub)menus and menu items will not be updated.

### Code examples:

Initialize the menu with the role option specified:

```
1 | $( ".selector" ).menu({ role: null });
```

Get or set the role option, after initialization:

```
1 | // getter
2 | var role = $( ".selector" ).menu( "option", "role"
3 |
4 | // setter
5 | $( ".selector" ).menu( "option", "role", null
```

## Methods

### blur( [event ] )

Removes focus from a menu, resets any active element styles and triggers the menu's `blur` event.

**event**

Type: [Event](#)

What triggered the menu to blur.

**Code examples:**

Invoke the blur method:

```
1 | $( ".selector" ).menu( "blur" );
```

### collapse( [event ] )

Closes the currently active sub-menu.

**event**

Type: [Event](#)

What triggered the menu to collapse.

**Code examples:**

Invoke the collapse method:

```
1 | $( ".selector" ).menu( "collapse" );
```

### collapseAll( [event ] [, all ] )

Closes all open sub-menus.

**event**

Type: [Event](#)

What triggered the menu to collapse.

**all**

Type: [Boolean](#)

Indicates whether all sub-menus should be closed or only sub-menus below and including the menu that is or contains the target of the triggering event.

**Code examples:**

Invoke the collapseAll method:

```
1 | $( ".selector" ).menu( "collapseAll", null, true
```

---

## destroy()

Removes the menu functionality completely. This will return the element back to its pre-init state.

This method does not accept any arguments.

**Code examples:**

Invoke the destroy method:

```
1 | $( ".selector" ).menu( "destroy" );
```

---

## disable()

Disables the menu.

This method does not accept any arguments.

### Code examples:

Invoke the disable method:

```
1 | $( ".selector" ).menu( "disable" );
```

---

## enable()

Enables the menu.

This method does not accept any arguments.

### Code examples:

Invoke the enable method:

```
1 | $( ".selector" ).menu( "enable" );
```

---

## expand( [event ] )

Opens the sub-menu below the currently active item, if one exists.

#### **event**

Type: [Event](#)

What triggered the menu to expand.

### Code examples:

Invoke the expand method:

```
1 | $( ".selector" ).menu( "expand" );
```

## focus( [event ], item )

Activates a particular menu item, begins opening any sub-menu if present and triggers the menu's `focus` event.

### event

Type: [Event](#)

What triggered the menu item to gain focus.

### item

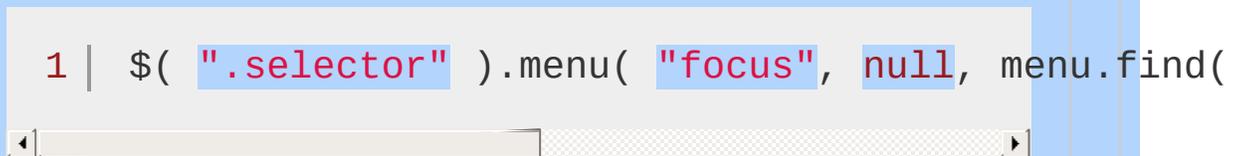
Type: [jQuery](#)

The menu item to focus/activate.

### Code examples:

Invoke the focus method:

```
1 | $( ".selector" ).menu( "focus", null, menu.find(
```



## isFirstItem()

Returns a boolean value stating whether or not the currently active item is the first item in the menu.

This method does not accept any arguments.

### Code examples:

Invoke the isFirstItem method:

```
1 | var firstItem = $( ".selector" ).menu( "isFirstItem"
```



## isLastItem()

Returns a boolean value stating whether or not the currently active item is the last item in the menu.

This method does not accept any arguments.

### Code examples:

Invoke the `isLastItem` method:

```
1 | var lastItem = $( ".selector" ).menu( "isLastItem"
```

---

## next( [event ] )

Moves active state to next menu item.

### event

Type: [Event](#)

What triggered the focus to move.

### Code examples:

Invoke the `next` method:

```
1 | $( ".selector" ).menu( "next" );
```

---

## nextPage( [event ] )

Moves active state to first menu item below the bottom of a scrollable menu or the last item if not scrollable.

### event

Type: [Event](#)

What triggered the focus to move.

### Code examples:

Invoke the `nextPage` method:

```
1 | $( ".selector" ).menu( "nextPage" );
```

## option( optionName )

Returns: *Object*

Gets the value currently associated with the specified `optionName`.

**optionName**

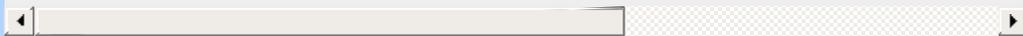
Type: *String*

The name of the option to get.

### Code examples:

Invoke the method:

```
1 | var isDisabled = $( ".selector" ).menu( "option"
```



## option()

Returns: *PlainObject*

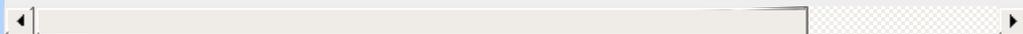
Gets an object containing key/value pairs representing the current menu options hash.

This method does not accept any arguments.

### Code examples:

Invoke the method:

```
1 | var options = $( ".selector" ).menu( "option"
```



## option( optionName, value )

Sets the value of the menu option associated with the specified `optionName`.

**optionName**

Type: String

The name of the option to set.

**value**

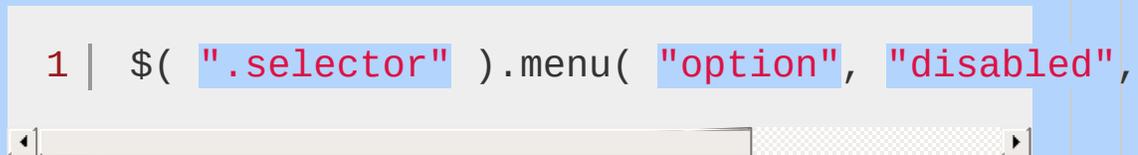
Type: Object

A value to set for the option.

**Code examples:**

Invoke the method:

```
1 | $( ".selector" ).menu( "option", "disabled",
```



---

## option( options )

Sets one or more options for the menu.

**options**

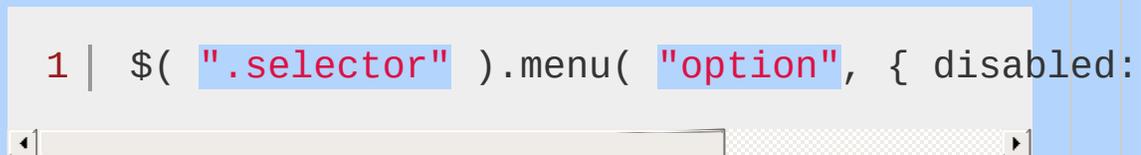
Type: Object

A map of option-value pairs to set.

**Code examples:**

Invoke the method:

```
1 | $( ".selector" ).menu( "option", { disabled:
```



---

## previous( [event ] )

Moves active state to previous menu item.

**event**

Type: [Event](#)  
What triggered the focus to move.

### Code examples:

Invoke the previous method:

```
1 | $( ".selector" ).menu( "previous" );
```

---

## previousPage( [event ] )

Moves active state to first menu item above the top of a scrollable menu or the first item if not scrollable.

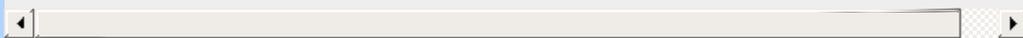
### event

Type: [Event](#)  
What triggered the focus to move.

### Code examples:

Invoke the previousPage method:

```
1 | $( ".selector" ).menu( "previousPage" );
```



---

## refresh()

Initializes sub-menus and menu items that have not already been initialized. New menu items, including sub-menus can be added to the menu or all of the contents of the menu can be replaced and then initialized with the `refresh()` method.

This method does not accept any arguments.

### Code examples:

Invoke the refresh method:

```
1 | $( ".selector" ).menu( "refresh" );
```

## select( [event ] )

Selects the currently active menu item, collapses all sub-menus and triggers the menu's `select` event.

### event

Type: `Event`

What triggered the selection.

### Code examples:

Invoke the select method:

```
1 | $( ".selector" ).menu( "select" );
```

## widget()

Returns: `jQuery`

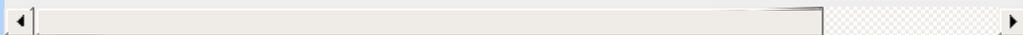
Returns a `jQuery` object containing the menu.

This method does not accept any arguments.

### Code examples:

Invoke the widget method:

```
1 | var widget = $( ".selector" ).menu( "widget" );
```



## Events

### blur( event, ui )

Type: `menublur`

Triggered when the menu loses focus.

**event**

Type: `Event`

**ui**

Type: `Object`

**item**

Type: `jQuery`

The currently active menu item.

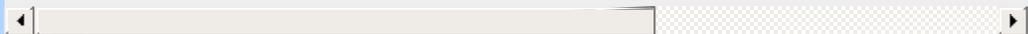
### Code examples:

Initialize the menu with the blur callback specified:

```
1 | $( ".selector" ).menu({  
2 |   blur: function( event, ui ) {}  
3 | });
```

Bind an event listener to the menublur event:

```
1 | $( ".selector" ).on( "menublur", function
```



### create( event, ui )

Type: `menucreate`

Triggered when the menu is created.

**event**  
Type: [Event](#)

**ui**  
Type: [Object](#)

### Code examples:

Initialize the menu with the create callback specified:

```
1 | $( ".selector" ).menu({  
2 |   create: function( event, ui ) {}  
3 | });
```

Bind an event listener to the menucreate event:

```
1 | $( ".selector" ).on( "menucreate", function( event, ui )
```



---

## focus( event, ui )

Type: `menufocus`

Triggered when a menu gains focus or when any menu item is activated.

**event**  
Type: [Event](#)

**ui**  
Type: [Object](#)

**item**  
Type: [jQuery](#)  
The currently active menu item.

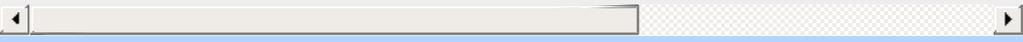
### Code examples:

Initialize the menu with the focus callback specified:

```
1 | $( ".selector" ).menu({
2 |   focus: function( event, ui ) {}
3 | });
```

Bind an event listener to the menufocus event:

```
1 | $( ".selector" ).on( "menufocus", function( event, ui
```



**select( event, ui )**

Type: `menuselect`

Triggered when a menu item is selected.

**event**

Type: [Event](#)

**ui**

Type: [Object](#)

**item**

Type: [jQuery](#)

The currently active menu item.

**Code examples:**

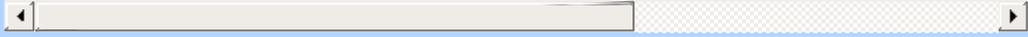
Initialize the menu with the select callback specified:

```
1 | $( ".selector" ).menu({
2 |   select: function( event, ui ) {}
3 | });
```

Bind an event listener to the menuselect event:

```
|
```

```
1 | $( ".selector" ).on( "menuselect", function( event, u
```



## Example:

### *A simple jQuery UI Menu*

```
1  <!doctype html>
2  <html lang="en">
3  <head>
4      <meta charset="utf-8">
5      <title>menu demo</title>
6      <link rel="stylesheet" href="http://code.jq
7      <style>
8      .ui-menu {
9          width: 200px;
10     }
11     </style>
12     <script src="http://code.jquery.com/jquery-:
13     <script src="http://code.jquery.com/ui/1.10
14 </head>
15 <body>
16
17 <ul id="menu">
18     <li><a href="#">Item 1</a></li>
19     <li><a href="#">Item 2</a></li>
20     <li><a href="#">Item 3</a>
21         <ul>
22             <li><a href="#">Item 3-1</a></li>
23             <li><a href="#">Item 3-2</a></li>
24             <li><a href="#">Item 3-3</a></li>
25             <li><a href="#">Item 3-4</a></li>
26             <li><a href="#">Item 3-5</a></li>
27         </ul>
28     </li>
29     <li><a href="#">Item 4</a></li>
30     <li><a href="#">Item 5</a></li>
31 </ul>
```

```
32
33 <script>
34 $( "#menu" ).menu();
35 </script>
36
37 </body>
38 </html>
```

Demo

POWERED BYHERONOTE

A new version of this book is available!

---

# Mouse Interaction

Categories: [Interactions](#) | [Utilities](#)

---

## Mouse Interaction

**Description:** *The base interaction layer.*

## QuickNav

### Options

[cancel](#)  
[delay](#)  
[distance](#)

### Methods

[\\_\\_mouseInit](#)  
[\\_\\_mouseDestroy](#)  
[\\_\\_mouseDown](#)  
[\\_\\_mouseMove](#)  
[\\_\\_mouseUp](#)  
[\\_\\_mouseDistanceMet](#)  
[\\_\\_mouseDelayMet](#)  
[\\_\\_mouseStart](#)  
[\\_\\_mouseDrag](#)  
[\\_\\_mouseStop](#)  
[\\_\\_mouseCapture](#)

### Events

Similar to [jQuery.Widget](#), the mouse interaction is not intended to be used directly. It is purely a base layer for other widgets to inherit from. This page only documents what is added to [jQuery.Widget](#), but it does include internal methods that are not intended to be overwritten. The intended public API is [\\_\\_mouseStart\(\)](#), [\\_\\_mouseDrag\(\)](#), [\\_\\_mouseStop\(\)](#), and [\\_\\_mouseCapture\(\)](#).

## Dependencies

[Widget Factory](#)

## Options

### cancel

Type: [Selector](#)

Default: `"input, textarea, button, select, option"`

Prevents interactions from starting on specified elements.

#### Code examples:

Initialize the mouse with the cancel option specified:

```
1 | $( ".selector" ).mouse({ cancel: ".title"
```

Get or set the cancel option, after initialization:

```
1 | // getter
2 | var cancel = $( ".selector" ).mouse( "op
3 |
4 | // setter
5 | $( ".selector" ).mouse( "option", "canc
```

### delay

Type: [Number](#)

Default: `0`

Time in milliseconds after mousedown until the interaction should start. This option can be used to prevent unwanted interactions when clicking on an element.

#### Code examples:

Initialize the mouse with the delay option specified:

```
1 | $( ".selector" ).mouse({ delay: 300 });
```

Get or set the delay option, after initialization:

```
1 | // getter
2 | var delay = $( ".selector" ).mouse( "option",
3 |
4 | // setter
5 | $( ".selector" ).mouse( "option", "delay", 300
```

## distance

Type: [Number](#)

Default: [1](#)

Distance in pixels after mousedown the mouse must move before the interaction should start. This option can be used to prevent unwanted interactions when clicking on an element.

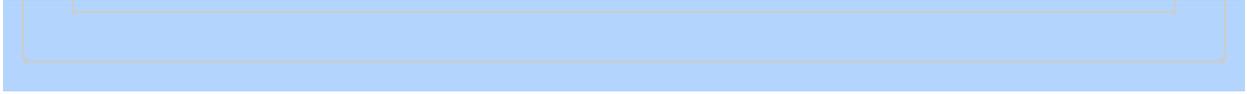
### Code examples:

Initialize the mouse with the distance option specified:

```
1 | $( ".selector" ).mouse({ distance: 10 });
```

Get or set the distance option, after initialization:

```
1 | // getter
2 | var distance = $( ".selector" ).mouse( "option"
3 |
4 | // setter
5 | $( ".selector" ).mouse( "option", "distance",
```



## Methods

### `_mouseCapture()`

Returns: *Boolean*

Determines whether an interaction should start based on event target of the interaction. The default implementation always returns `true`.

This method does not accept any arguments.

#### Code examples:

Invoke the `_mouseCapture` method:

```
1 | $( ".selector" ).mouse( "_mouseCapture"
```

### `_mouseDelayMet()`

Returns: *Boolean*

Determines whether the `delay` option has been met for the current interaction.

This method does not accept any arguments.

#### Code examples:

Invoke the `_mouseDelayMet` method:

```
1 | $( ".selector" ).mouse( "_mouseDelayMet"
```

### `_mouseDestroy()`

Destroys the interaction event handlers. This must be called from the extending widget's `_destroy()` method.

This method does not accept any arguments.

### Code examples:

Invoke the `_mouseDestroy` method:

```
1 | $( ".selector" ).mouse( "_mouseDestroy" );
```

---

## `_mouseDistanceMet()`

Returns: *Boolean*

Determines whether the `distance` option has been met for the current interaction.

This method does not accept any arguments.

### Code examples:

Invoke the `_mouseDistanceMet` method:

```
1 | $( ".selector" ).mouse( "_mouseDistanceMet" );
```

---

## `_mouseDown()`

Handles the beginning of an interaction. Verifies that the event is associated with the primary mouse button and ensures that the `delay` and `distance` options are met prior to starting the interaction. When the interaction is ready to start, invokes the `_mouseStart()` method for the extending widget to handle.

This method does not accept any arguments.

### Code examples:

Invoke the `_mouseDown` method:

```
1 | $( ".selector" ).mouse( "_mouseDown" );
```

## `_mouseDrag()`

The extending widget should implement a `_mouseDrag()` method to handle each movement of an interaction. This method will receive the mouse event associated with the movement.

This method does not accept any arguments.

### Code examples:

Invoke the `_mouseDrag` method:

```
1 | $( ".selector" ).mouse( "_mouseDrag" );
```

## `_mouseInit()`

Initializes the interaction event handlers. This must be called from the extending widget's `_create()` method.

This method does not accept any arguments.

### Code examples:

Invoke the `_mouseInit` method:

```
1 | $( ".selector" ).mouse( "_mouseInit" );
```

## `_mouseMove()`

Handles each movement of the interaction. Invokes the `mouseDrag()` method for the extending widget to handle.

This method does not accept any arguments.

### Code examples:

Invoke the `_mouseMove` method:

```
1 | $( ".selector" ).mouse( "_mouseMove" );
```

## `_mouseStart()`

The extending widget should implement a `_mouseStart()` method to handle the beginning of an interaction. This method will receive the mouse event associated with the start of the interaction.

This method does not accept any arguments.

### Code examples:

Invoke the `_mouseStart` method:

```
1 | $( ".selector" ).mouse( "_mouseStart" );
```

## `_mouseStop()`

The extending widget should implement a `_mouseStop()` method to handle the end of an interaction. This method will

receive the mouse event associated with the end of the interaction.

This method does not accept any arguments.

### Code examples:

Invoke the `_mouseStop` method:

```
1 | $( ".selector" ).mouse( "_mouseStop" );
```

## `_mouseUp()`

Handles the end of the interaction. Invokes the `mouseStop()` method for the extending widget to handle.

This method does not accept any arguments.

### Code examples:

Invoke the `_mouseUp` method:

```
1 | $( ".selector" ).mouse( "_mouseUp" );
```

POWERED BYHERONOTE

A new version of this book is available!

---

# .position()

Categories: [Method Overrides](#) | [Methods](#) | [Utilities](#)

`.position( options )`

Returns: [jQuery](#)  
version added: 1.8

**Description:** *Position an element relative to another.*

## `.position( options )`

### options

Type: [Object](#)

**my** (default: `"center"`)

Type: [String](#)

Defines which position **on the element being positioned** to align with the target element: "horizontal vertical" alignment. A single value such as `"right"` will be normalized to `"right center"`, `"top"` will be normalized to `"center top"` (following CSS convention). Acceptable horizontal values: `"left"`, `"center"`, `"right"`. Acceptable vertical values: `"top"`, `"center"`, `"bottom"`. Example: `"left top"` or `"center center"`. Each dimension can also contain offsets, in pixels or percent, e.g., `"right+10 top-25%"`. Percentage offsets are relative to the element being positioned.

**at** (default: `"center"`)

Type: [String](#)

Defines which position **on the target element** to align the positioned element against: "horizontal vertical" alignment. See the `my` option for full details on possible values. Percentage offsets are relative to the target element.

**of** (default: `null`)

Type: [Selector](#) or [Element](#) or [jQuery](#) or [Event](#)

Which element to position against. If you provide a selector or jQuery object, the first matching element will be used. If you provide an event object, the `pageX` and `pageY` properties will be used. Example: `"#top-menu"`

**collision** (default: `"flip"`)

Type: [String](#)

When the positioned element overflows the window in

some direction, move it to an alternative position. Similar to `my` and `at`, this accepts a single value or a pair for horizontal/vertical, e.g., `"flip"`, `"fit"`, `"fit flip"`, `"fit none"`.

`"flip"`: Flips the element to the opposite side of the target and the collision detection is run again to see if it will fit. Whichever side allows more of the element to be visible will be used.

`"fit"`: Shift the element away from the edge of the window.

`"flipfit"`: First applies the flip logic, placing the element on whichever side allows more of the element to be visible. Then the fit logic is applied to ensure as much of the element is visible as possible.

`"none"`: Does not apply any collision detection.

**using** (default: `null`)

Type: `Function()`

When specified, the actual property setting is delegated to this callback. Receives two parameters: The first is a hash of `top` and `left` values for the position that should be set and can be forwarded to `.css()` or `.animate()`.

The second provides feedback about the position and dimensions of both elements, as well as calculations to their relative position. Both `target` and `element` have these properties: `element`, `left`, `top`, `width`, `height`. In addition, there's `horizontal`, `vertical` and `important`, giving you twelve potential directions like `{ horizontal: "center", vertical: "left", important: "horizontal" }`.

**within** (default: `window`)

Type: `Selector` or `Element` or `jQuery`

Element to position within, affecting collision detection. If you provide a selector or jQuery object, the first matching element will be used.

The jQuery UI `.position()` method allows you to position an element

relative to the window, document, another element, or the cursor/mouse, without worrying about offset parents.

*Note: jQuery UI does not support positioning hidden elements.*

This is a standalone jQuery plugin and has no dependencies on other jQuery UI components.

This plugin extends jQuery's built-in `.position()` method. If jQuery UI is not loaded, calling the `.position()` method may not fail directly, as the method still exists. However, the expected behavior will not occur.

## Example:

*A simple jQuery UI Position example.*

```
1  <!doctype html>
2  <html lang="en">
3  <head>
4      <meta charset="utf-8">
5      <title>position demo</title>
6      <link rel="stylesheet" href="http://code.jq
7      <style>
8          .positionDiv {
9              position: absolute;
10             width: 75px;
11             height: 75px;
12             background: green;
13         }
14     </style>
15     <script src="http://code.jquery.com/jquery-:
16     <script src="http://code.jquery.com/ui/1.10
17 </head>
18 <body>
19
20 <div id="targetElement">
21     <div class="positionDiv" id="position1"></d.
22     <div class="positionDiv" id="position2"></d.
23     <div class="positionDiv" id="position3"></d.
24     <div class="positionDiv" id="position4"></d.
25 </div>
26
27 <script>
28 $( "#position1" ).position({
29     my: "center",
30     at: "center",
31     of: "#targetElement"
```

```
32 });
33
34 $( "#position2" ).position({
35   my: "left top",
36   at: "left top",
37   of: "#targetElement"
38 });
39
40 $( "#position3" ).position({
41   my: "right center",
42   at: "right bottom",
43   of: "#targetElement"
44 });
45
46 $( document ).mousemove(function( event ) {
47   $( "#position4" ).position({
48     my: "left+3 bottom-3",
49     of: event,
50     collision: "fit"
51   });
52 });
53 </script>
54
55 </body>
56 </html>
```

Demo

**A new version of this book is available!**

---

# Progressbar Widget

Categories: [Widgets](#)

---

**Progressbar Widget**

version added: 1.6

**Description:** *Display status of a determinate or indeterminate process.*

## QuickNav

## Examples

### Options

[disabled](#)  
[max](#)  
[value](#)

### Methods

[destroy](#)  
[disable](#)  
[enable](#)  
[option](#)  
[value](#)  
[widget](#)

### Events

[change](#)  
[create](#)  
[complete](#)

The progress bar is designed to display the current percent complete for a process. The bar is coded to be flexibly sized through CSS and will scale to fit inside its parent container by default.

A determinate progress bar should only be used in situations where the system can accurately update the current status. A determinate progress bar should never fill from left to right, then loop back to empty for a single process — if the actual status cannot be calculated, an indeterminate progress bar should be used to provide user feedback.

## Dependencies

[UI Core](#)

[Widget Factory](#)

## Additional Notes:

This widget requires some functional CSS, otherwise it won't work. If you build a custom theme, use the widget's specific CSS file as a starting point.

## Options

### disabled

Type: [Boolean](#)

Default: `false`

Disables the progressbar if set to `true`.

#### Code examples:

Initialize the progressbar with the disabled option specified:

```
1 | $( ".selector" ).progressbar({ disabled
```



Get or set the disabled option, after initialization:

```
1 | // getter
2 | var disabled = $( ".selector" ).progress
3 |
4 | // setter
5 | $( ".selector" ).progressbar( "option",
```



### max

Type: [Number](#)

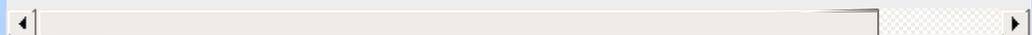
Default: `100`

The maximum value of the progressbar.

#### Code examples:

Initialize the progressbar with the max option specified:

```
1 | $( ".selector" ).progressbar({ max: 1024
```



Get or set the max option, after initialization:

```
1 // getter
2 var max = $( ".selector" ).progressbar( "option"
3
4 // setter
5 $( ".selector" ).progressbar( "option", "max"
```

**value**

**Type:** [Number](#) or [Boolean](#)

**Default:** 0

The value of the progressbar.

**Multiple types supported:**

**Number:** A value between 0 and the `max`.

**Boolean:** Value can be set to `false` to create an indeterminate progressbar.

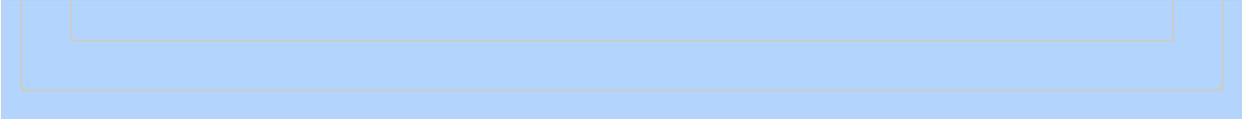
**Code examples:**

Initialize the progressbar with the value option specified:

```
1 $( ".selector" ).progressbar({ value: 25 });
```

Get or set the value option, after initialization:

```
1 // getter
2 var value = $( ".selector" ).progressbar( "option"
3
4 // setter
5 $( ".selector" ).progressbar( "option", "value"
```



## Methods

### destroy()

Removes the progressbar functionality completely. This will return the element back to its pre-init state.

This method does not accept any arguments.

#### Code examples:

Invoke the destroy method:

```
1 | $( ".selector" ).progressbar( "destroy"
```

### disable()

Disables the progressbar.

This method does not accept any arguments.

#### Code examples:

Invoke the disable method:

```
1 | $( ".selector" ).progressbar( "disable"
```

### enable()

Enables the progressbar.

This method does not accept any arguments.

### Code examples:

Invoke the enable method:

```
1 | $( ".selector" ).progressbar( "enable" );
```

## option( optionName )

Returns: *Object*

Gets the value currently associated with the specified `optionName`.

**optionName**

Type: *String*

The name of the option to get.

### Code examples:

Invoke the method:

```
1 | var isDisabled = $( ".selector" ).progressbar(
```

## option()

Returns: *PlainObject*

Gets an object containing key/value pairs representing the current progressbar options hash.

This method does not accept any arguments.

### Code examples:

Invoke the method:

```
1 | var options = $( ".selector" ).progressbar( "option"
```

## option( optionName, value )

Sets the value of the progressbar option associated with the specified `optionName`.

### **optionName**

Type: String

The name of the option to set.

### **value**

Type: Object

A value to set for the option.

### **Code examples:**

Invoke the method:

```
1 | $( ".selector" ).progressbar( "option", "disabled"
```

## option( options )

Sets one or more options for the progressbar.

### **options**

Type: Object

A map of option-value pairs to set.

### **Code examples:**

Invoke the method:

```
1 | $( ".selector" ).progressbar( "option", { disabled:
```

## value()

Returns: [Number](#) or [Boolean](#)

Gets the current value of the progressbar.

This method does not accept any arguments.

### Code examples:

Invoke the method:

```
1 | var progressSoFar = $( ".selector" ).progressbar(
```



## value( value )

Sets the current value of the progressbar.

### value

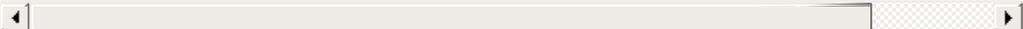
Type: [Number](#) or [Boolean](#)

The value to set. See the [value](#) option for details on valid values.

### Code examples:

Invoke the method:

```
1 | $( ".selector" ).progressbar( "value", 50 );
```



## widget()

Returns: [jQuery](#)

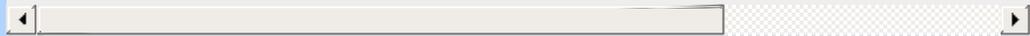
Returns a [jQuery](#) object containing the progressbar.

This method does not accept any arguments.

### Code examples:

Invoke the widget method:

```
1 | var widget = $( ".selector" ).progressbar( "widget"
```



## Events

**change( event, ui )**      *Type:* progressbarchange

Triggered when the value of the progressbar changes.

**event**  
Type: [Event](#)

**ui**  
Type: [Object](#)

### Code examples:

Initialize the progressbar with the change callback specified:

```
1 | $( ".selector" ).progressbar({  
2 |   change: function( event, ui ) {}  
3 | });
```

Bind an event listener to the progressbarchange event:

```
1 | $( ".selector" ).on( "progressbarchange"
```



**complete( event, ui )**      *Type:* progressbarcomplete

Triggered when the value of the progressbar reaches the maximum value.

**event**  
Type: [Event](#)

**ui**

Type: [Object](#)

### Code examples:

Initialize the progressbar with the complete callback specified:

```
1 | $( ".selector" ).progressbar({
2 |   complete: function( event, ui ) {}
3 | });
```

Bind an event listener to the progressbarcomplete event:

```
1 | $( ".selector" ).on( "progressbarcomplete", function
```



**create( event, ui )**

Type: [progressbarcreate](#)

Triggered when the progressbar is created.

**event**

Type: [Event](#)

**ui**

Type: [Object](#)

### Code examples:

Initialize the progressbar with the create callback specified:

```
1 | $( ".selector" ).progressbar({
2 |   create: function( event, ui ) {}
3 | });
```

Bind an event listener to the progressbarcreate event:

```
1 | $( ".selector" ).on( "progressbarcreate", function
```

## Examples:

### Example: *A simple jQuery UI Progressbar*

```
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>progressbar demo</title>
6   <link rel="stylesheet" href="http://code.jq
7   <script src="http://code.jquery.com/jquery-:
8   <script src="http://code.jquery.com/ui/1.10
9 </head>
10 <body>
11
12 <div id="progressbar"></div>
13
14 <script>
15 $( "#progressbar" ).progressbar({
16   value: 37
17 });
18 </script>
19
20 </body>
21 </html>
```

Demo

### Example: *A simple jQuery UI Indeterminate Progressbar*

```
1 <!doctype html>
2 <html lang="en">
3 <head>
```

```
4 <meta charset="utf-8">
5 <title>progressbar demo</title>
6 <link rel="stylesheet" href="http://code.jquery.com/u
7 <script src="http://code.jquery.com/jquery-1.9.1.js"
8 <script src="http://code.jquery.com/ui/1.10.1/jquery-
9 </head>
10 <body>
11
12 <div id="progressbar"></div>
13
14 <script>
15 $( "#progressbar" ).progressbar({
16     value: false
17 });
18 </script>
19
20 </body>
21 </html>
```

Demo

POWERED BYHERONOTE

A new version of this book is available!

---

# Puff Effect

Categories: [Effects](#)

## Puff Effect

**Description:** *Creates a puff effect by scaling the element up and hiding it at the same time.*

### puff

**percent** (default: 150)

Type: Number

The percentage to scale to.

## Example:

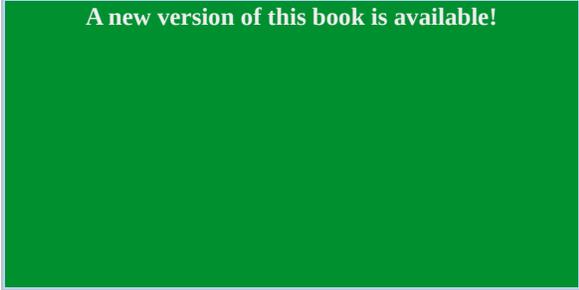
*Toggle a div using the puff effect.*

```
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>puff demo</title>
6   <link rel="stylesheet" href="http://code.jq
7   <style>
8     #toggle {
9       width: 100px;
10      height: 100px;
11      background: #ccc;
12    }
13  </style>
14  <script src="http://code.jquery.com/jquery-:
15  <script src="http://code.jquery.com/ui/1.10
16 </head>
17 <body>
18
19 <p>Click anywhere to toggle the box.</p>
20 <div id="toggle"></div>
21
22 <script>
23 $( document ).click(function() {
24   $( "#toggle" ).toggle( "puff" );
25 });
26 </script>
27
28 </body>
29 </html>
```

Demo

POWERED BYHERONOTE

A new version of this book is available!



---

# Pulsate Effect

Categories: [Effects](#)

## Pulsate Effect

**Description:** *The pulsate effect hides or shows an element by pulsing it in or out.*

### pulsate

**times** (default: 5)

Type: Integer

The number of times the element should pulse. An extra half pulse is added for hide/show.

## Example:

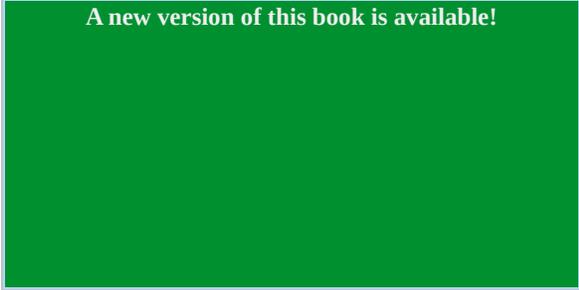
*Toggle a div using the pulsate effect.*

```
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>pulsate demo</title>
6   <link rel="stylesheet" href="http://code.jq
7   <style>
8     #toggle {
9       width: 100px;
10      height: 100px;
11      background: #ccc;
12    }
13  </style>
14  <script src="http://code.jquery.com/jquery-:
15  <script src="http://code.jquery.com/ui/1.10
16 </head>
17 <body>
18
19 <p>Click anywhere to toggle the box.</p>
20 <div id="toggle"></div>
21
22 <script>
23 $( document ).click(function() {
24   $( "#toggle" ).toggle( "pulsate" );
25 });
26 </script>
27
28 </body>
29 </html>
```

Demo

POWERED BYHERONOTE

A new version of this book is available!



---

# .removeClass()

Categories: [Effects](#) | [Effects Core](#) | [Method Overrides](#)

```
.removeClass( className [, duration ] [, easing ] [, complete ] ) Returns: jQuery
```

**Description:** *Removes the specified class(es) from each of the set of matched elements while animating all style changes.*

```
.removeClass( className [, duration ] [, easing ] [, complete ] )
```

**className**

Type: [String](#)

One or more class names (space separated) to be removed from the class attribute of each matched element.

**duration** (default: `400`)

Type: [Number](#) or [String](#)

A string or number determining how long the animation will run.

**easing** (default: `swing`)

Type: [String](#)

A string indicating which [easing](#) function to use for the transition.

**complete**

Type: [Function\(\)](#)

A function to call once the animation is complete.

Similar to native CSS transitions, jQuery UI's class animations provide a smooth transition from one state to another while allowing you to keep all the details about which styles to change in CSS and out of your JavaScript. All class animation methods, including `.removeClass()`, support custom durations and easings, as well as providing a callback for when the animation completes.

Not all styles can be animated. For example, there is no way to animate a background image. Any styles that cannot be animated will be changed at the end of the animation.

This plugin extends jQuery's built-in `.removeClass()` method. If

jQuery UI is not loaded, calling the `.removeClass()` method may not fail directly, as the method still exists. However, the expected behavior will not occur.

## Example:

*Removes the class "big-blue" from the matched elements.*

```
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>removeClass demo</title>
6   <link rel="stylesheet" href="http://code.jq
7   <style>
8   div {
9     width: 100px;
10    height: 100px;
11    background-color: #ccc;
12  }
13  .big-blue {
14    width: 200px;
15    height: 200px;
16    background-color: #00f;
17  }
18  </style>
19  <script src="http://code.jquery.com/jquery-:
20  <script src="http://code.jquery.com/ui/1.10
21 </head>
22 <body>
23
24 <div class="big-blue"></div>
25
26 <script>
27 $( "div" ).click(function() {
28   $( this ).removeClass( "big-blue", 1000, "e
29 });
30 </script>
```

```
31  
32 </body>  
33 </html>
```

Demo

POWERED BYHERONOTE

A new version of this book is available!

---

# .removeUniqueId()

Categories: [Methods](#) | [UI Core](#)

`.removeUniqueId()`

Returns: *jQuery*

**Description:** *Remove ids that were set by `.uniqueId()` for the set of matched elements.*

**`.removeUniqueId()`**

version added: 1.9

This method does not accept any arguments.

The `.removeUniqueId()` will remove ids that were set by `.uniqueId()`. Calling `.removeUniqueId()` on an element that did not have its id set by `.uniqueId()` will have no affect, even if the element has an id.

**POWERED BYHERONOTE**

A new version of this book is available!

---

# Resizable Widget

Categories: [Interactions](#)

---

**Resizable Widget**

version added: 1.0

**Description:** *Change the size of an element using the mouse.*

## QuickNav

## Examples

### Options

[alsoResize](#)  
[animate](#)  
[animateDuration](#)  
[animateEasing](#)  
[aspectRatio](#)  
[autoHide](#)  
[cancel](#)  
[containment](#)  
[delay](#)  
[disabled](#)  
[distance](#)  
[ghost](#)  
[grid](#)  
[handles](#)  
[helper](#)  
[maxHeight](#)  
[maxWidth](#)  
[minHeight](#)  
[minWidth](#)

### Methods

[destroy](#)  
[disable](#)  
[enable](#)  
[option](#)  
[widget](#)

### Events

[create](#)  
[start](#)  
[resize](#)  
[stop](#)

The jQuery UI Resizable plugin makes selected elements resizable (meaning they have draggable resize handles). You can specify one or more handles as well as min and max width and height.

## Dependencies

[UI Core](#)

[Widget Factory](#)

[Mouse Interaction](#)

## Additional Notes:

This widget requires some functional CSS, otherwise it won't work. If you build a custom theme, use the widget's specific CSS file as a starting point.

## Options

**alsoResize** Type: [Selector](#) or [jQuery](#) or [Element](#)

Default: `false`

One or more elements to resize synchronously with the resizable element.

### Code examples:

Initialize the resizable with the alsoResize option specified:

```
1 | $( ".selector" ).resizable({ alsoResize
```

Get or set the alsoResize option, after initialization:

```
1 | // getter
2 | var alsoResize = $( ".selector" ).resizable
3 |
4 | // setter
5 | $( ".selector" ).resizable( "option", "a
```

## animate

Type: [Boolean](#)

Default: `false`

Animates to the final size after resizing.

### Code examples:

Initialize the resizable with the animate option specified:

```
1 | $( ".selector" ).resizable({ animate: t
```

Get or set the animate option, after initialization:

```
1 // getter
2 var animate = $( ".selector" ).resizable( "option"
3
4 // setter
5 $( ".selector" ).resizable( "option", "animate"
```

## animateDuration

Type: [Number](#) or [String](#)

Default: "slow"

How long to animate when using the [animate](#) option.

**Multiple types supported:**

**Number:** Duration in milliseconds.

**String:** A named duration, such as "slow" or "fast".

### Code examples:

Initialize the resizable with the animateDuration option specified:

```
1 $( ".selector" ).resizable({ animateDuration:
```

Get or set the animateDuration option, after initialization:

```
1 // getter
2 var animateDuration = $( ".selector" ).resizable(
3
4 // setter
5 $( ".selector" ).resizable( "option", "animateDuration"
```

## animateEasing

Type: String

Default: "swing"

Which easing to apply when using the animate option.

### Code examples:

Initialize the resizable with the animateEasing option specified:

```
1 | $( ".selector" ).resizable({ animateEasing: "easeOutB"
◀ |
▶
```

Get or set the animateEasing option, after initialization:

```
1 | // getter
2 | var animateEasing = $( ".selector" ).resizable(
3 |
4 | // setter
5 | $( ".selector" ).resizable( "option", "animateEasing"
◀ |
▶
```

## aspectRatio

Type: Boolean or Number

Default: false

Whether the element should be constrained to a specific aspect ratio.

### Multiple types supported:

**Boolean:** When set to true, the element will maintain its original aspect ratio.

**Number:** Force the element to maintain a specific aspect ratio during resizing.

### Code examples:

Initialize the resizable with the aspectRatio option specified:

```
1 | $( ".selector" ).resizable({ aspectRatio: true
```

Get or set the aspectRatio option, after initialization:

```
1 | // getter
2 | var aspectRatio = $( ".selector" ).resizable(
3 |
4 | // setter
5 | $( ".selector" ).resizable( "option", "aspectRatio"
```

## autoHide

Type: [Boolean](#)

Default: `false`

Whether the handles should hide when the user is not hovering over the element.

### Code examples:

Initialize the resizable with the autoHide option specified:

```
1 | $( ".selector" ).resizable({ autoHide: true });
```

Get or set the autoHide option, after initialization:

```
1 | // getter
2 | var autoHide = $( ".selector" ).resizable( "option"
3 |
4 | // setter
5 | $( ".selector" ).resizable( "option", "autoHide"
```

## cancel

Type: [Selector](#)

Default: `"input, textarea, button, select, option"`

Prevents resizing from starting on specified elements.

### Code examples:

Initialize the resizable with the cancel option specified:

```
1 | $( ".selector" ).resizable({ cancel: ".cancel"
```

Get or set the cancel option, after initialization:

```
1 | // getter
2 | var cancel = $( ".selector" ).resizable( "option"
3 |
4 | // setter
5 | $( ".selector" ).resizable( "option", "cancel"
```

## containment Type: [Selector](#) or [Element](#) or [String](#)

Default: `false`

Constrains resizing to within the bounds of the specified element or region.

### Multiple types supported:

**Selector:** The resizable element will be contained to the bounding box of the first element found by the selector. If no element is found, no containment will be set.

**Element:** The resizable element will be contained to the bounding box of this element.

**String:** Possible values: `"parent"` and `"document"`.

### Code examples:

Initialize the resizable with the containment option specified:

```
1 | $( ".selector" ).resizable({ containment: "parent"
```

Get or set the containment option, after initialization:

```
1 | // getter
2 | var containment = $( ".selector" ).resizable(
3 |
4 | // setter
5 | $( ".selector" ).resizable( "option", "containment"
```

## delay

Type: [Number](#)

Default: `0`

Tolerance, in milliseconds, for when resizing should start. If specified, resizing will not start until after mouse is moved beyond duration. This can help prevent unintended resizing when clicking on an element.

### Code examples:

Initialize the resizable with the delay option specified:

```
1 | $( ".selector" ).resizable({ delay: 150 });
```

Get or set the delay option, after initialization:

```
1 | // getter
2 | var delay = $( ".selector" ).resizable( "option"
3 |
4 | // setter
```

```
5 | $( ".selector" ).resizable( "option", "delay"
```

## disabled

Type: Boolean

Default: `false`

Disables the resizable if set to `true`.

### Code examples:

Initialize the resizable with the disabled option specified:

```
1 | $( ".selector" ).resizable({ disabled: true });
```

Get or set the disabled option, after initialization:

```
1 | // getter
2 | var disabled = $( ".selector" ).resizable( "option"
3 |
4 | // setter
5 | $( ".selector" ).resizable( "option", "disabled"
```

## distance

Type: Number

Default: `1`

Tolerance, in pixels, for when resizing should start. If specified, resizing will not start until after mouse is moved beyond distance. This can help prevent unintended resizing when clicking on an element.

### Code examples:

Initialize the resizable with the distance option specified:

```
1 | $( ".selector" ).resizable({ distance: 30 });
```

Get or set the distance option, after initialization:

```
1 | // getter
2 | var distance = $( ".selector" ).resizable( "option"
3 |
4 | // setter
5 | $( ".selector" ).resizable( "option", "distance"
```

## ghost

Type: [Boolean](#)

Default: `false`

If set to `true`, a semi-transparent helper element is shown for resizing.

**Code examples:**

Initialize the resizable with the ghost option specified:

```
1 | $( ".selector" ).resizable({ ghost: true });
```

Get or set the ghost option, after initialization:

```
1 | // getter
2 | var ghost = $( ".selector" ).resizable( "option"
3 |
4 | // setter
5 | $( ".selector" ).resizable( "option", "ghost"
```

## grid

Type: [Array](#)

Default: `false`

Snaps the resizing element to a grid, every x and y pixels.

Array values: `[ x, y ]`.

### Code examples:

Initialize the resizable with the grid option specified:

```
1 | $( ".selector" ).resizable({ grid: [ 20, 10 ] });
```

Get or set the grid option, after initialization:

```
1 | // getter
2 | var grid = $( ".selector" ).resizable( "option"
3 |
4 | // setter
5 | $( ".selector" ).resizable( "option", "grid", [
```

## handles

Type: [String](#) or [Object](#)

Default: `"e, s, se"`

Which handles can be used for resizing.

### Multiple types supported:

**String:** A comma delimited list of any of the following: n, e, s, w, ne, se, sw, nw, all. The necessary handles will be auto-generated by the plugin.

### Object:

The following keys are supported: { n, e, s, w, ne, se, sw, nw }. The value of any specified should be a jQuery selector matching the child element of the resizable to use as that handle. If the handle is not a child of the resizable, you can pass in the DOMElement or a valid jQuery object directly.

Note: When generating your own handles, each handle must have the `ui-resizable-handle` class, as well as the appropriate `ui-resizable-{direction}` class, .e.g., `ui-resizable-s`.

### Code examples:

Initialize the resizable with the handles option specified:

```
1 | $( ".selector" ).resizable({ handles: "n, e, s, w"
```

Get or set the handles option, after initialization:

```
1 | // getter
2 | var handles = $( ".selector" ).resizable( "option"
3 |
4 | // setter
5 | $( ".selector" ).resizable( "option", "handles"
```

## helper

Type: [String](#)

Default: `false`

A class name that will be added to a proxy element to outline the resize during the drag of the resize handle. Once the resize is complete, the original element is sized.

### Code examples:

Initialize the resizable with the helper option specified:

```
1 | $( ".selector" ).resizable({ helper: "resizable-helper"
```

Get or set the helper option, after initialization:

```
1 // getter
2 var helper = $( ".selector" ).resizable( "option"
3
4 // setter
5 $( ".selector" ).resizable( "option", "helper"
```

## maxHeight

Type: [Number](#)

Default: `null`

The maximum height the resizable should be allowed to resize to.

### Code examples:

Initialize the resizable with the maxHeight option specified:

```
1 $( ".selector" ).resizable({ maxHeight: 300 });
```

Get or set the maxHeight option, after initialization:

```
1 // getter
2 var maxHeight = $( ".selector" ).resizable( "option"
3
4 // setter
5 $( ".selector" ).resizable( "option", "maxHeight"
```

## maxWidth

Type: [Number](#)

Default: `null`

The maximum width the resizable should be allowed to resize to.

### Code examples:

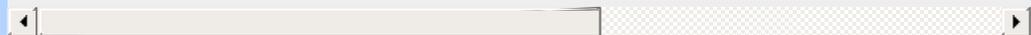
Initialize the resizable with the `maxWidth` option specified:

```
1 | $( ".selector" ).resizable({ maxWidth: 300 });
```



Get or set the `maxWidth` option, after initialization:

```
1 | // getter
2 | var maxWidth = $( ".selector" ).resizable( "option"
3 |
4 | // setter
5 | $( ".selector" ).resizable( "option", "maxWidth"
```



## minHeight

Type: [Number](#)

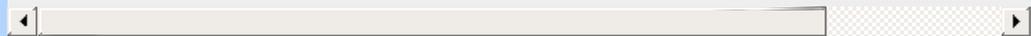
Default: `10`

The minimum height the resizable should be allowed to resize to.

### Code examples:

Initialize the resizable with the `minHeight` option specified:

```
1 | $( ".selector" ).resizable({ minHeight: 150 });
```



Get or set the `minHeight` option, after initialization:

```
1 | // getter
2 | var minHeight = $( ".selector" ).resizable( "option"
3 |
4 | // setter
5 | $( ".selector" ).resizable( "option", "minHeight"
```

## minWidth

Type: [Number](#)

Default: `10`

The minimum width the resizable should be allowed to resize to.

### Code examples:

Initialize the resizable with the minWidth option specified:

```
1 | $( ".selector" ).resizable({ minWidth: 150 });
```

Get or set the minWidth option, after initialization:

```
1 | // getter
2 | var minWidth = $( ".selector" ).resizable( "option"
3 |
4 | // setter
5 | $( ".selector" ).resizable( "option", "minWidth"
```

## Methods

### destroy()

Removes the resizable functionality completely. This will return the element back to its pre-init state.

This method does not accept any arguments.

#### Code examples:

Invoke the destroy method:

```
1 | $( ".selector" ).resizable( "destroy" )
```

### disable()

Disables the resizable.

This method does not accept any arguments.

#### Code examples:

Invoke the disable method:

```
1 | $( ".selector" ).resizable( "disable" )
```

### enable()

Enables the resizable.

This method does not accept any arguments.

### Code examples:

Invoke the enable method:

```
1 | $( ".selector" ).resizable( "enable" );
```

## option( optionName )

Returns: *Object*

Gets the value currently associated with the specified `optionName`.

**optionName**

Type: *String*

The name of the option to get.

### Code examples:

Invoke the method:

```
1 | var isDisabled = $( ".selector" ).resizable(
```

## option()

Returns: *PlainObject*

Gets an object containing key/value pairs representing the current resizable options hash.

This method does not accept any arguments.

### Code examples:

Invoke the method:

```
1 | var options = $( ".selector" ).resizable( "option"
```

## option( optionName, value )

Sets the value of the resizable option associated with the specified `optionName`.

### **optionName**

Type: String

The name of the option to set.

### **value**

Type: Object

A value to set for the option.

### **Code examples:**

Invoke the method:

```
1 | $( ".selector" ).resizable( "option", "disabled"
```

## option( options )

Sets one or more options for the resizable.

### **options**

Type: Object

A map of option-value pairs to set.

### **Code examples:**

Invoke the method:

```
1 | $( ".selector" ).resizable( "option", { disabled:
```

## widget()

Returns: *jQuery*

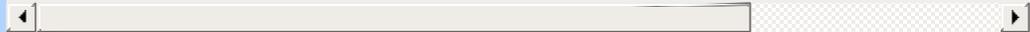
Returns a `jQuery` object containing the resizable element.

This method does not accept any arguments.

### Code examples:

Invoke the `widget` method:

```
1 | var widget = $( ".selector" ).resizable( "widget"
```



## Events

**create( event, ui )**

Type: `resizecreate`

Triggered when the resizable is created.

**event**

Type: [Event](#)

**ui**

Type: [Object](#)

### Code examples:

Initialize the resizable with the create callback specified:

```
1 | $( ".selector" ).resizable({  
2 |   create: function( event, ui ) {}  
3 | });
```

Bind an event listener to the resizecreate event:

```
1 | $( ".selector" ).on( "resizecreate", fun  
◀ | _____ ▶
```

**resize( event, ui )**

Type: `resize`

This event is triggered during the resize, on the drag of the resize handler.

**event**

Type: [Event](#)

## ui

Type: [Object](#)

### element

Type: [jQuery](#)

The jQuery object representing the element to be resized

### helper

Type: [jQuery](#)

The jQuery object representing the helper that's being resized

### originalElement

Type: [jQuery](#)

The jQuery object representing the original element before it is wrapped

### originalPosition

Type: [Object](#)

The position represented as `{ left, top }` before the resizable is resized

### originalSize

Type: [Object](#)

The size represented as `{ width, height }` before the resizable is resized

### position

Type: [Object](#)

The current position represented as `{ left, top }`

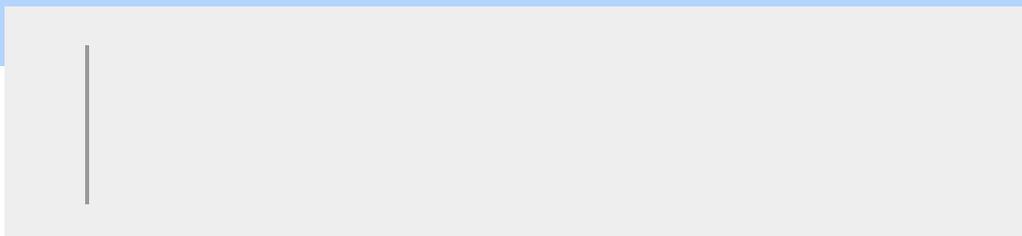
### size

Type: [Object](#)

The current size represented as `{ width, height }`

## Code examples:

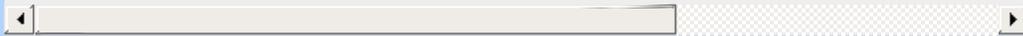
Initialize the resizable with the resize callback specified:



```
1 | $( ".selector" ).resizable({
2 |   resize: function( event, ui ) {}
3 | });
```

Bind an event listener to the resize event:

```
1 | $( ".selector" ).on( "resize", function( event, ui )
```



**start( event, ui )**

*Type:* `resizestart`

This event is triggered at the start of a resize operation.

**event**

Type: [Event](#)

**ui**

Type: [Object](#)

**element**

Type: [jQuery](#)

The jQuery object representing the element to be resized

**helper**

Type: [jQuery](#)

The jQuery object representing the helper that's being resized

**originalElement**

Type: [jQuery](#)

The jQuery object representing the original element before it is wrapped

**originalPosition**

Type: [Object](#)

The position represented as `{ left, top }` before

the resizable is resized

### originalSize

Type: Object

The size represented as `{ width, height }` before the resizable is resized

### position

Type: Object

The current position represented as `{ left, top }`

### size

Type: Object

The current size represented as `{ width, height }`

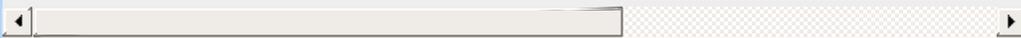
## Code examples:

Initialize the resizable with the start callback specified:

```
1 | $( ".selector" ).resizable({  
2 |   start: function( event, ui ) {}  
3 | });
```

Bind an event listener to the `resizestart` event:

```
1 | $( ".selector" ).on( "resizestart", function( event,
```



## stop( event, ui )

Type: `resizestop`

This event is triggered at the end of a resize operation.

### event

Type: Event

## ui

Type: [Object](#)

### element

Type: [jQuery](#)

The jQuery object representing the element to be resized

### helper

Type: [jQuery](#)

The jQuery object representing the helper that's being resized

### originalElement

Type: [jQuery](#)

The jQuery object representing the original element before it is wrapped

### originalPosition

Type: [Object](#)

The position represented as `{ left, top }` before the resizable is resized

### originalSize

Type: [Object](#)

The size represented as `{ width, height }` before the resizable is resized

### position

Type: [Object](#)

The current position represented as `{ left, top }`

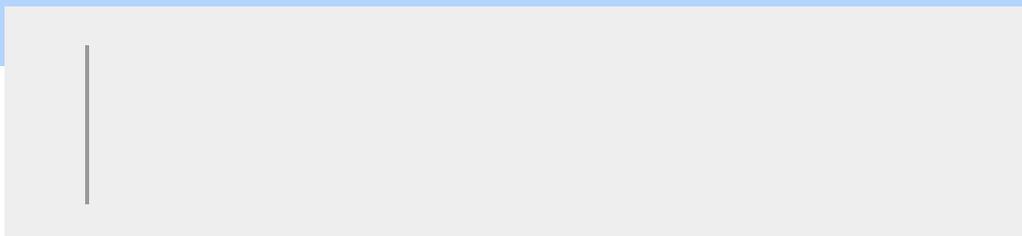
### size

Type: [Object](#)

The current size represented as `{ width, height }`

## Code examples:

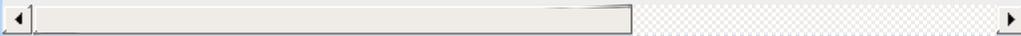
Initialize the resizable with the stop callback specified:



```
1 | $( ".selector" ).resizable({  
2 |   stop: function( event, ui ) {}  
3 | });
```

Bind an event listener to the `resizestop` event:

```
1 | $( ".selector" ).on( "resizestop", function( event, ui ) {}
```



## Example:

*A simple jQuery UI Resizable.*

```
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>resizable demo</title>
6   <link rel="stylesheet" href="http://code.jq
7   <style>
8     #resizable {
9       width: 100px;
10      height: 100px;
11      background: #ccc;
12    } </style>
13    <script src="http://code.jquery.com/jquery-:
14    <script src="http://code.jquery.com/ui/1.10
15  </head>
16  <body>
17
18  <div id="resizable"></div>
19
20  <script>
21  $( "#resizable" ).resizable();
22  </script>
23
24  </body>
25  </html>
```

Demo

**A new version of this book is available!**

---

# Scale Effect

Categories: [Effects](#)

## Scale Effect

**Description:** *Shrink or grow an element by a percentage factor.*

### scale

**direction** (default: `"both"`)

Type: String

The direction of the effect. Possible values: `"both"`, `"vertical"` or `"horizontal"`.

**origin** (default: [ `"middle"`, `"center"` ])

Type: Array

The vanishing point.

**percent**

Type: Number

The percentage to scale to.

**scale** (default: `"both"`)

Type: String

Which areas of the element will be resized: `"both"`, `"box"`, `"content"`. `Box` resizes the border and padding of the element; `content` resizes any content inside of the element.

## Examples:

**Example:** *Toggle a div using the scale effect.*

```
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>scale demo</title>
6   <link rel="stylesheet" href="http://code.jq
7   <style>
8   #toggle {
9     width: 100px;
10    height: 100px;
11    background: #ccc;
12  }
13 </style>
14 <script src="http://code.jquery.com/jquery-:
15 <script src="http://code.jquery.com/ui/1.10
16 </head>
17 <body>
18
19 <p>Click anywhere to toggle the box.</p>
20 <div id="toggle"></div>
21
22 <script>
23 $( document ).click(function() {
24   $( "#toggle" ).toggle( "scale" );
25 });
26 </script>
27
28 </body>
29 </html>
```

## Demo

**Example** Toggle a div using the scale effect in just one direction.

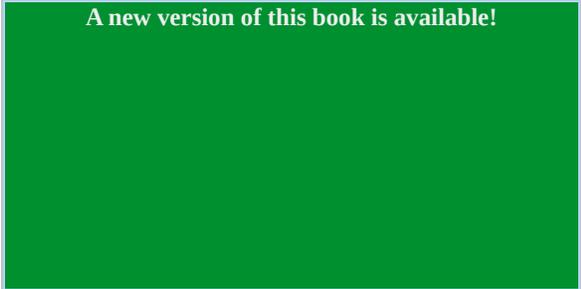
```
1  <!doctype html>
2  <html lang="en">
3  <head>
4      <meta charset="utf-8">
5      <title>scale demo</title>
6      <link rel="stylesheet" href="http://code.jq
7      <style>
8          #toggle {
9              width: 100px;
10             height: 100px;
11             background: #ccc;
12         }
13     </style>
14     <script src="http://code.jquery.com/jquery-:
15     <script src="http://code.jquery.com/ui/1.10
16 </head>
17 <body>
18
19 <p>Click anywhere to toggle the box.</p>
20 <div id="toggle"></div>
21
22 <script>
23 $( document ).click(function() {
24     $( "#toggle" ).toggle({ effect: "scale", di
25 });
26 </script>
27
28 </body>
29 </html>
```

## Demo

---

POWERED BYHERONOTE

A new version of this book is available!



---

# `.scrollParent()`

Categories: [Methods](#) | [UI Core](#)

`.scrollTop()`

Returns: `jQuery`

**Description:** *Get the closest ancestor element that is scrollable.*

**`.scrollTop()`**

This method does not accept any arguments.

This method finds the nearest ancestor that allows scrolling. In other words, the `.scrollTop()` method finds the element that the currently selected element will scroll within.

*Note: This method only works on jQuery objects containing one element.*

**POWERED BYHERONOTE**

A new version of this book is available!

---

# Selectable Widget

Categories: [Interactions](#)

---

Selectable Widget

version added: 1.0

**Description:** *Use the mouse to select elements, individually or in a group.*

## QuickNav

## Examples

### Options

[appendTo](#)  
[autoRefresh](#)  
[cancel](#)  
[delay](#)  
[disabled](#)  
[distance](#)  
[filter](#)  
[tolerance](#)

### Methods

[destroy](#)  
[disable](#)  
[enable](#)  
[refresh](#)  
[option](#)  
[widget](#)

### Events

[create](#)  
[selected](#)  
[selecting](#)  
[start](#)  
[stop](#)  
[unselected](#)  
[unselecting](#)

The jQuery UI Selectable plugin allows for elements to be selected by dragging a box (sometimes called a lasso) with the mouse over the elements. Elements can also be selected via click or drag while holding the ctrl/meta key, allowing for multiple (non-contiguous) selections.

## Dependencies

[UI Core](#)

[Widget Factory](#)

[Mouse Interaction](#)

## Additional Notes:

This widget requires some functional CSS, otherwise it won't work. If you build a custom theme, use the widget's specific CSS file as a starting point.

## Options

### appendTo

Type: [Selector](#)

Default: `"body"`

Which element the selection helper (the lasso) should be appended to.

#### Code examples:

Initialize the selectable with the appendTo option specified:

```
1 | $( ".selector" ).selectable({ appendTo:
```

Get or set the appendTo option, after initialization:

```
1 | // getter
2 | var appendTo = $( ".selector" ).selectable
3 |
4 | // setter
5 | $( ".selector" ).selectable( "option", "
```

### autoRefresh

Type: [Boolean](#)

Default: `true`

This determines whether to refresh (recalculate) the position and size of each selectee at the beginning of each select operation. If you have many items, you may want to set this to false and call the `refresh()` method manually.

#### Code examples:

Initialize the selectable with the autoRefresh option specified:

```
1 | $( ".selector" ).selectable({ autoRefresh: false
```

Get or set the autoRefresh option, after initialization:

```
1 | // getter
2 | var autoRefresh = $( ".selector" ).selectable(
3 |
4 | // setter
5 | $( ".selector" ).selectable( "option", "autoRefresh"
```

## cancel

Type: [Selector](#)

Default: "input, textarea, button, select, option"

Prevents selecting if you start on elements matching the selector.

### Code examples:

Initialize the selectable with the cancel option specified:

```
1 | $( ".selector" ).selectable({ cancel: "a, .cancel"
```

Get or set the cancel option, after initialization:

```
1 | // getter
2 | var cancel = $( ".selector" ).selectable( "option"
3 |
4 | // setter
5 | $( ".selector" ).selectable( "option", "cancel"
```

## delay

Type: [Integer](#)

Default: `0`

Time in milliseconds to define when the selecting should start. This helps prevent unwanted selections when clicking on an element.

### Code examples:

Initialize the selectable with the delay option specified:

```
1 | $( ".selector" ).selectable({ delay: 150 });
```

Get or set the delay option, after initialization:

```
1 | // getter
2 | var delay = $( ".selector" ).selectable( "option"
3 |
4 | // setter
5 | $( ".selector" ).selectable( "option", "delay"
```

## disabled

Type: [Boolean](#)

Default: `false`

Disables the selectable if set to `true`.

### Code examples:

Initialize the selectable with the disabled option specified:

```
1 | $( ".selector" ).selectable({ disabled: true });
```

Get or set the disabled option, after initialization:

```
1 | // getter
2 | var disabled = $( ".selector" ).selectable( "option"
3 |
4 | // setter
5 | $( ".selector" ).selectable( "option", "disabled"
```

## distance

Type: [Number](#)

Default: `0`

Tolerance, in pixels, for when selecting should start. If specified, selecting will not start until the mouse has been dragged beyond the specified distance.

### Code examples:

Initialize the selectable with the distance option specified:

```
1 | $( ".selector" ).selectable({ distance: 30 });
```

Get or set the distance option, after initialization:

```
1 | // getter
2 | var distance = $( ".selector" ).selectable( "option"
3 |
4 | // setter
5 | $( ".selector" ).selectable( "option", "distance"
```

## filter

Type: [Selector](#)

Default: `"*"`

The matching child elements will be made selectees (able to

be selected).

### Code examples:

Initialize the selectable with the filter option specified:

```
1 | $( ".selector" ).selectable({ filter: "li" });
```

Get or set the filter option, after initialization:

```
1 | // getter
2 | var filter = $( ".selector" ).selectable( "option"
3 |
4 | // setter
5 | $( ".selector" ).selectable( "option", "filter"
```

## tolerance

Type: [String](#)

Default: "touch"

Specifies which mode to use for testing whether the lasso should select an item. Possible values:

"fit": Lasso overlaps the item entirely.

"touch": Lasso overlaps the item by any amount.

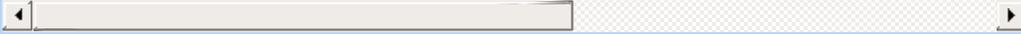
### Code examples:

Initialize the selectable with the tolerance option specified:

```
1 | $( ".selector" ).selectable({ tolerance: "fit"
```

Get or set the tolerance option, after initialization:

```
1 // getter
2 var tolerance = $( ".selector" ).selectable(
3
4 // setter
5 $( ".selector" ).selectable( "option", "tolerance"
```



## Methods

### destroy()

Removes the selectable functionality completely. This will return the element back to its pre-init state.

This method does not accept any arguments.

#### Code examples:

Invoke the destroy method:

```
1 | $( ".selector" ).selectable( "destroy"
```

### disable()

Disables the selectable.

This method does not accept any arguments.

#### Code examples:

Invoke the disable method:

```
1 | $( ".selector" ).selectable( "disable"
```

### enable()

Enables the selectable.

This method does not accept any arguments.

### Code examples:

Invoke the enable method:

```
1 | $( ".selector" ).selectable( "enable" );
```

## option( optionName )

Returns: *Object*

Gets the value currently associated with the specified `optionName`.

**optionName**

Type: *String*

The name of the option to get.

### Code examples:

Invoke the method:

```
1 | var isDisabled = $( ".selector" ).selectable(
```

## option()

Returns: *PlainObject*

Gets an object containing key/value pairs representing the current selectable options hash.

This method does not accept any arguments.

### Code examples:

Invoke the method:

```
1 | var options = $( ".selector" ).selectable( "option"
```

## option( optionName, value )

Sets the value of the selectable option associated with the specified `optionName`.

### **optionName**

Type: String

The name of the option to set.

### **value**

Type: Object

A value to set for the option.

### **Code examples:**

Invoke the method:

```
1 | $( ".selector" ).selectable( "option", "disabled"
```

## option( options )

Sets one or more options for the selectable.

### **options**

Type: Object

A map of option-value pairs to set.

### **Code examples:**

Invoke the method:

```
1 | $( ".selector" ).selectable( "option", { disabled:
```

## refresh()

Refresh the position and size of each selectee element. This method can be used to manually recalculate the position and size of each selectee when the `autoRefresh` option is set to `false`.

This method does not accept any arguments.

### Code examples:

Invoke the refresh method:

```
1 | $( ".selector" ).selectable( "refresh" );
```

## widget()

Returns: *jQuery*

Returns a `jQuery` object containing the selectable element.

This method does not accept any arguments.

### Code examples:

Invoke the widget method:

```
1 | var widget = $( ".selector" ).selectable( "widget" );
```

## Events

**create( event, ui )**      *Type:* `selectablecreate`

Triggered when the selectable is created.

**event**  
Type: [Event](#)

**ui**  
Type: [Object](#)

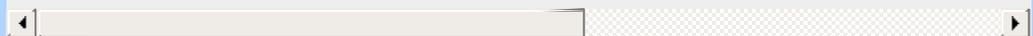
### Code examples:

Initialize the selectable with the create callback specified:

```
1 | $( ".selector" ).selectable({  
2 |   create: function( event, ui ) {}  
3 | });
```

Bind an event listener to the selectablecreate event:

```
1 | $( ".selector" ).on( "selectablecreate",
```



**selected( event, ui )**      *Type:* `selectableselected`

Triggered at the end of the select operation, on each element added to the selection.

**event**  
Type: [Event](#)

**ui**

Type: Object

**selected**

Type: Element

The selectable item that has been selected.

### Code examples:

Initialize the selectable with the selected callback specified:

```
1 | $( ".selector" ).selectable({  
2 |   selected: function( event, ui ) {}  
3 | });
```

Bind an event listener to the selectableselected event:

```
1 | $( ".selector" ).on( "selectableselected", function
```

---

**selecting( event, ui )**    *Type:* selectableselecting

Triggered during the select operation, on each element added to the selection.

**event**

Type: Event

**ui**

Type: Object

**selecting**

Type: Element

The current selectable item being selected.

### Code examples:

Initialize the selectable with the selecting callback specified:

```
1 | $( ".selector" ).selectable({
2 |   selecting: function( event, ui ) {}
3 | });
```

Bind an event listener to the selectableselecting event:

```
1 | $( ".selector" ).on( "selectableselecting", function
```

**start( event, ui )**

Type: `selectablestart`

Triggered at the beginning of the select operation.

**event**

Type: `Event`

**ui**

Type: `Object`

**Code examples:**

Initialize the selectable with the start callback specified:

```
1 | $( ".selector" ).selectable({
2 |   start: function( event, ui ) {}
3 | });
```

Bind an event listener to the selectablestart event:

```
1 | $( ".selector" ).on( "selectablestart", function
```

## stop( event, ui )

Type: `selectablestop`

Triggered at the end of the select operation.

**event**

Type: `Event`

**ui**

Type: `Object`

### Code examples:

Initialize the selectable with the stop callback specified:

```
1 | $( ".selector" ).selectable({
2 |   stop: function( event, ui ) {}
3 | });
```

Bind an event listener to the selectablestop event:

```
1 | $( ".selector" ).on( "selectablestop", function
```



## unselected( event, ui )

Type: `selectableunselected`

Triggered at the end of the select operation, on each element removed from the selection.

**event**

Type: `Event`

**ui**

Type: `Object`



specified:

```
1 | $( ".selector" ).selectable({  
2 |   unselecting: function( event, ui ) {}  
3 | });
```

Bind an event listener to the selectableunselecting event:

```
1 | $( ".selector" ).on( "selectableunselecting",
```

## Example:

### *A simple jQuery UI Selectable.*

```
1  <!doctype html>
2  <html lang="en">
3  <head>
4    <meta charset="utf-8">
5    <title>selectable demo</title>
6    <link rel="stylesheet" href="http://code.jqu
7    <style>
8    #selectable .ui-selecting {
9      background: #ccc;
10   }
11   #selectable .ui-selected {
12     background: #999;
13   }
14   </style>
15   <script src="http://code.jquery.com/jquery-:
16   <script src="http://code.jquery.com/ui/1.10
17 </head>
18 <body>
19
20 <ul id="selectable">
21   <li>Item 1</li>
22   <li>Item 2</li>
23   <li>Item 3</li>
24   <li>Item 4</li>
25   <li>Item 5</li>
26 </ul>
27
28 <script>
29 $( "#selectable" ).selectable();
30 </script>
31
```

```
32 | </body>  
33 | </html>
```

Demo

**POWERED BYHERONOTE**

**A new version of this book is available!**

---

# Shake Effect

Categories: [Effects](#)

## Shake Effect

**Description:** *Shakes the element multiple times, vertically or horizontally.*

### shake

**direction** (default: `"both"`)

Type: String

The direction of the effect. Possible values: `"left"`, `"right"`, `"up"`, `"down"`.

**distance** (default: `20`)

Type: Number

Distance to shake.

**times** (default: `3`)

Type: Integer

Times to shake.

## Example:

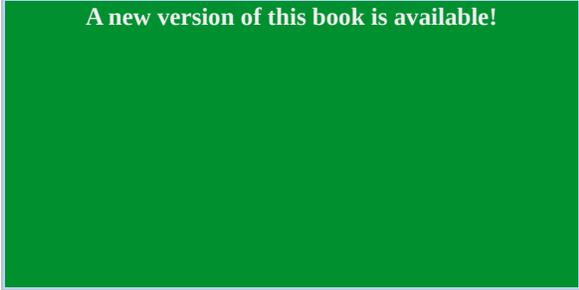
### *Shake a div.*

```
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>shake demo</title>
6   <link rel="stylesheet" href="http://code.jq
7   <style>
8     #toggle {
9       width: 100px;
10      height: 100px;
11      background: #ccc;
12    }
13  </style>
14  <script src="http://code.jquery.com/jquery-:
15  <script src="http://code.jquery.com/ui/1.10
16 </head>
17 <body>
18
19 <p>Click anywhere to shake the box.</p>
20 <div id="toggle"></div>
21
22 <script>
23 $( document ).click(function() {
24   $( "#toggle" ).effect( "shake" );
25 });
26 </script>
27
28 </body>
29 </html>
```

Demo

POWERED BYHERONOTE

A new version of this book is available!



---

# .show()

Categories: [Effects](#) | [Effects Core](#) | [Method Overrides](#) | [Methods](#)

```
.show( effect [, options ] [, duration ] [, complete ] )
```

Returns: [jQuery](#)

**Description:** *Display the matched elements, using custom effects.*

```
.show( effect [, options ] [, duration ] [, complete ] )
```

#### **effect**

Type: [String](#)

A string indicating which [effect](#) to use for the transition.

#### **options**

Type: [Object](#)

Effect-specific settings and [easing](#).

#### **duration** (default: `400`)

Type: [Number](#) or [String](#)

A string or number determining how long the animation will run.

#### **complete**

Type: [Function\(\)](#)

A function to call once the animation is complete.

```
.show( options )
```

#### **options**

Type: [Object](#)

All animation settings. The only required property is `effect`.

#### **effect**

Type: [String](#)

A string indicating which [effect](#) to use for the transition.

#### **easing** (default: `"swing"`)

Type: [String](#)

A string indicating which [easing](#) function to use for the transition.

#### **duration** (default: `400`)

Type: [Number](#) or [String](#)

A string or number determining how long the animation will run.

**complete**

Type: `Function()`

A function to call once the animation is complete.

This plugin extends jQuery's built-in `.show()` method. If jQuery UI is not loaded, calling the `.show()` method may not fail directly, as the method still exists. However, the expected behavior will not occur.

## Example:

*Show a div using the fold effect.*

```
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>show demo</title>
6   <link rel="stylesheet" href="http://code.jq
7   <style>
8   div {
9     display: none;
10    width: 100px;
11    height: 100px;
12    background: #ccc;
13    border: 1px solid #000;
14  }
15 </style>
16 <script src="http://code.jquery.com/jquery-:
17 <script src="http://code.jquery.com/ui/1.10
18 </head>
19 <body>
20
21 <button>show the div</button>
22 <div></div>
23
24 <script>
25 $( "button" ).click(function() {
26   $( "div" ).show( "fold", 1000 );
27 });
28 </script>
29
30 </body>
31 </html>
```

Demo

POWERED BYHERONOTE

A new version of this book is available!

---

# Size Effect

Categories: [Effects](#)

## Size Effect

**Description:** *Resize an element to a specified width and height.*

### size

#### to

Type: Object

Height and width to resize to.

**origin** (default: [ "top", "left" ])

Type: Array

The vanishing point.

**scale** (default: "both")

Type: String

Which areas of the element will be resized: "both", "box", "content". Box resizes the border and padding of the element; content resizes any content inside of the element.

## Example:

*Resize the element using the size effect.*

```
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>size demo</title>
6   <link rel="stylesheet" href="http://code.jq
7   <style>
8     #toggle {
9       width: 100px;
10      height: 100px;
11      background: #ccc;
12    }
13  </style>
14  <script src="http://code.jquery.com/jquery-:
15  <script src="http://code.jquery.com/ui/1.10
16 </head>
17 <body>
18
19 <p>Click anywhere to resize the box.</p>
20 <div id="toggle"></div>
21
22 <script>
23 $( document ).click(function() {
24   $( "#toggle" ).effect( "size", {
25     to: { width: 200, height: 60 }
26   }, 1000 );
27 });
28 </script>
29
30 </body>
31 </html>
```

Demo

POWERED BYHERONOTE

A new version of this book is available!

---

# Slide Effect

Categories: [Effects](#)

## Slide Effect

**Description:** *Slides the element out of the viewport.*

### slide

**direction** (default: "both")

Type: String

The direction of the effect. Possible values: "left", "right", "up", "down".

**distance** (default: element's outerWidth)

Type: Number

The distance of the effect. Defaults to either the height or width of the element depending on the `direction` argument. Can be set to any integer less than the width/height of the element.

## Example:

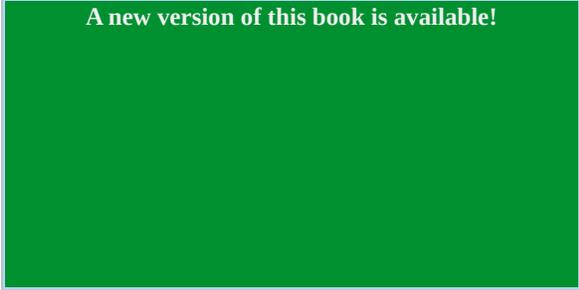
*Toggle a div using the slide effect.*

```
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>slide demo</title>
6   <link rel="stylesheet" href="http://code.jq
7   <style>
8     #toggle {
9       width: 100px;
10      height: 100px;
11      background: #ccc;
12    }
13  </style>
14  <script src="http://code.jquery.com/jquery-:
15  <script src="http://code.jquery.com/ui/1.10
16 </head>
17 <body>
18
19 <p>Click anywhere to toggle the box.</p>
20 <div id="toggle"></div>
21
22 <script>
23 $( document ).click(function() {
24   $( "#toggle" ).toggle( "slide" );
25 });
26 </script>
27
28 </body>
29 </html>
```

Demo

POWERED BYHERONOTE

A new version of this book is available!



---

# Slider Widget

Categories: [Widgets](#)

---

Slider Widget

version added: 1.5

**Description:** *Drag a handle to select a numeric value.*

## QuickNav

## Examples

### Options

[animate](#)  
[disabled](#)  
[max](#)  
[min](#)  
[orientation](#)  
[range](#)  
[step](#)  
[value](#)  
[values](#)

### Methods

[destroy](#)  
[disable](#)  
[enable](#)  
[option](#)  
[value](#)  
[values](#)  
[widget](#)

### Events

[create](#)  
[start](#)  
[slide](#)  
[change](#)  
[stop](#)

The jQuery UI Slider plugin makes selected elements into sliders. There are various options such as multiple handles and ranges. The handle can be moved with the mouse or the arrow keys.

The slider widget will create handle elements with the class `ui-slider-handle` on initialization. You can specify custom handle elements by creating and appending the elements and adding the `ui-slider-handle` class before initialization. It will only create the number of handles needed to match the length of `value/values`. For example, if you specify `values: [ 1, 5, 18 ]` and create one custom handle, the plugin will create the other two.

## Dependencies

[UI Core](#)

[Widget Factory](#)

[Mouse Interaction](#)

## Additional Notes:

This widget requires some functional CSS, otherwise it won't work. If you build a custom theme, use the widget's specific

CSS file as a starting point.



## disabled

Type: [Boolean](#)

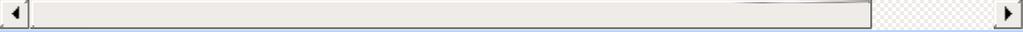
Default: `false`

Disables the slider if set to `true`.

### Code examples:

Initialize the slider with the disabled option specified:

```
1 | $( ".selector" ).slider({ disabled: true });
```



Get or set the disabled option, after initialization:

```
1 | // getter
2 | var disabled = $( ".selector" ).slider( "option"
3 |
4 | // setter
5 | $( ".selector" ).slider( "option", "disabled"
```



## max

Type: [Number](#)

Default: `100`

The maximum value of the slider.

### Code examples:

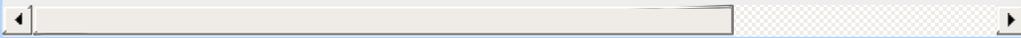
Initialize the slider with the max option specified:

```
1 | $( ".selector" ).slider({ max: 50 });
```

Get or set the max option, after initialization:

```
1 | // getter
2 | var max = $( ".selector" ).slider( "option",
```

```
3
4 // setter
5 $( ".selector" ).slider( "option", "max", 50 );
```



## min

Type: [Number](#)

Default: `0`

The minimum value of the slider.

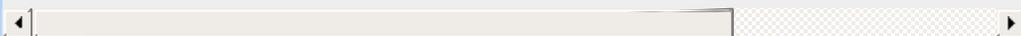
### Code examples:

Initialize the slider with the min option specified:

```
1 $( ".selector" ).slider({ min: 10 });
```

Get or set the min option, after initialization:

```
1 // getter
2 var min = $( ".selector" ).slider( "option",
3
4 // setter
5 $( ".selector" ).slider( "option", "min", 10 );
```



## orientation

Type: [String](#)

Default: `"horizontal"`

Determines whether the slider handles move horizontally (min on left, max on right) or vertically (min on bottom, max on top). Possible values: `"horizontal"`, `"vertical"`.

### Code examples:

Initialize the slider with the orientation option specified:

```
|
```

```
1 | $( ".selector" ).slider({ orientation: "vertical"
```

Get or set the orientation option, after initialization:

```
1 | // getter
2 | var orientation = $( ".selector" ).slider( "option"
3 |
4 | // setter
5 | $( ".selector" ).slider( "option", "orientation"
```

## range

Type: Boolean or String

Default: `false`

Whether the slider represents a range.

**Multiple types supported:**

**Boolean:** If set to `true`, the slider will detect if you have two handles and create a stylable range element between these two.

**String:** Either `"min"` or `"max"`. A min range goes from the slider min to one handle. A max range goes from one handle to the slider max.

**Code examples:**

Initialize the slider with the range option specified:

```
1 | $( ".selector" ).slider({ range: true });
```

Get or set the range option, after initialization:

```
1 | // getter
```

```
2 var range = $( ".selector" ).slider( "option"  
3  
4 // setter  
5 $( ".selector" ).slider( "option", "range", true
```

## step

Type: [Number](#)

Default: [1](#)

Determines the size or amount of each interval or step the slider takes between the min and max. The full specified value range of the slider (max - min) should be evenly divisible by the step.

### Code examples:

Initialize the slider with the step option specified:

```
1 | $( ".selector" ).slider({ step: 5 });
```

Get or set the step option, after initialization:

```
1 // getter  
2 var step = $( ".selector" ).slider( "option",  
3  
4 // setter  
5 $( ".selector" ).slider( "option", "step", 5 );
```

## value

Type: [Number](#)

Default: [0](#)

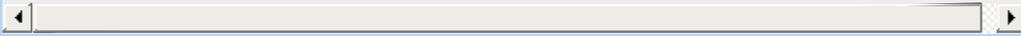
Determines the value of the slider, if there's only one handle. If there is more than one handle, determines the value of the

first handle.

### Code examples:

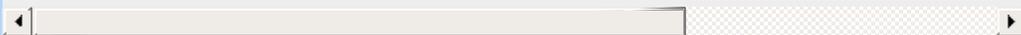
Initialize the slider with the value option specified:

```
1 | $( ".selector" ).slider({ value: 10 });
```



Get or set the value option, after initialization:

```
1 | // getter
2 | var value = $( ".selector" ).slider( "option"
3 |
4 | // setter
5 | $( ".selector" ).slider( "option", "value", 10
```



## values

Type: [Array](#)

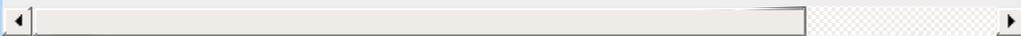
Default: `null`

This option can be used to specify multiple handles. If the `range` option is set to `true`, the length of `values` should be 2.

### Code examples:

Initialize the slider with the values option specified:

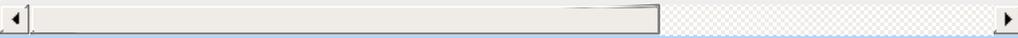
```
1 | $( ".selector" ).slider({ values: [ 10, 25 ] });
```



Get or set the values option, after initialization:

```
1 | // getter
2 | var values = $( ".selector" ).slider( "option"
3 |
4 | // setter
```

```
5 $( ".selector" ).slider( "option", "values", [
```



## Methods

### destroy()

Removes the slider functionality completely. This will return the element back to its pre-init state.

This method does not accept any arguments.

#### Code examples:

Invoke the destroy method:

```
1 | $( ".selector" ).slider( "destroy" );
```

### disable()

Disables the slider.

This method does not accept any arguments.

#### Code examples:

Invoke the disable method:

```
1 | $( ".selector" ).slider( "disable" );
```

### enable()

Enables the slider.

This method does not accept any arguments.

### Code examples:

Invoke the enable method:

```
1 | $( ".selector" ).slider( "enable" );
```

## option( optionName )

Returns: *Object*

Gets the value currently associated with the specified `optionName`.

**optionName**

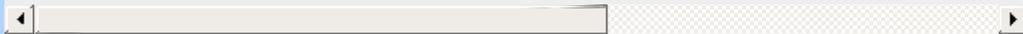
Type: *String*

The name of the option to get.

### Code examples:

Invoke the method:

```
1 | var isDisabled = $( ".selector" ).slider( "option"
```



## option()

Returns: *PlainObject*

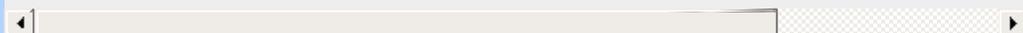
Gets an object containing key/value pairs representing the current slider options hash.

This method does not accept any arguments.

### Code examples:

Invoke the method:

```
1 | var options = $( ".selector" ).slider( "option"
```



## option( optionName, value )

Sets the value of the slider option associated with the specified `optionName`.

**optionName**

Type: String

The name of the option to set.

**value**

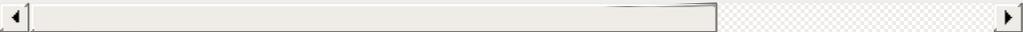
Type: Object

A value to set for the option.

**Code examples:**

Invoke the method:

```
1 | $( ".selector" ).slider( "option", "disabled"
```



## option( options )

Sets one or more options for the slider.

**options**

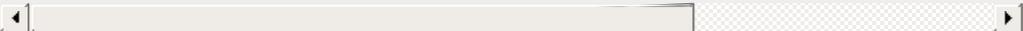
Type: Object

A map of option-value pairs to set.

**Code examples:**

Invoke the method:

```
1 | $( ".selector" ).slider( "option", { disabled:
```



## value()

Returns: *Number*

Get the value of the slider.

This method does not accept any arguments.

### Code examples:

Invoke the method:

```
1 | var selection = $( ".selector" ).slider( "value"
```



## value( value )

Set the value of the slider.

**value**

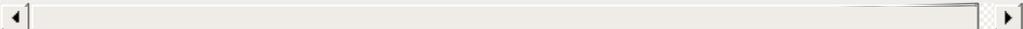
Type: *Number*

The value to set.

### Code examples:

Invoke the method:

```
1 | $( ".selector" ).slider( "value", 55 );
```



## values()

Returns: *Array*

Get the value for all handles.

This method does not accept any arguments.

### Code examples:

Invoke the method:

```
1 | var values = $( ".selector" ).slider( "values"
```

## values( index )

Returns: *Number*

Get the value for the specified handle.

### index

Type: *Integer*

The zero-based index of the handle.

### Code examples:

Invoke the method:

```
1 | var value = $( ".selector" ).slider( "values"
```

## values( index, value )

Set the value for the specified handle.

### index

Type: *Integer*

The zero-based index of the handle.

### value

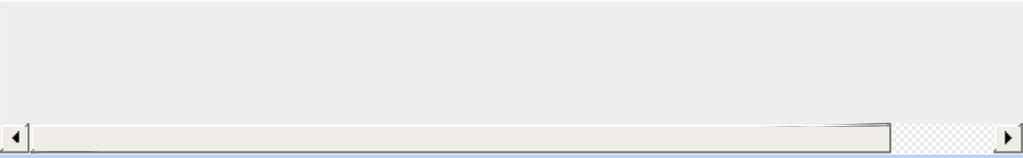
Type: *Number*

The value to set.

### Code examples:

Invoke the method:

```
1 | $( ".selector" ).slider( "values", 0, 55 );
```



---

## values( values )

Set the value for all handles.

### values

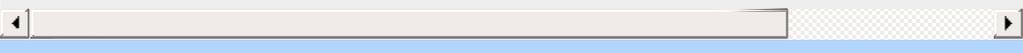
Type: [Array](#)

The values to set.

### Code examples:

Invoke the method:

```
1 | $( ".selector" ).slider( "values", [ 55, 105 ] );
```



---

## widget()

Returns: [jQuery](#)

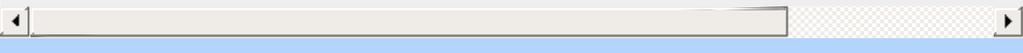
Returns a [jQuery](#) object containing the slider.

This method does not accept any arguments.

### Code examples:

Invoke the widget method:

```
1 | var widget = $( ".selector" ).slider( "widget" );
```



## Events

### change( event, ui )

Type: `slidechange`

Triggered after the user slides a handle, if the value has changed; or if the value is changed programmatically via the `value` method.

#### event

Type: `Event`

#### ui

Type: `Object`

#### handle

Type: `jQuery`

The jQuery object representing the handle that was changed.

#### value

Type: `Number`

The current value of the slider.

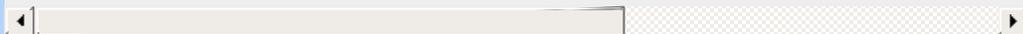
### Code examples:

Initialize the slider with the change callback specified:

```
1 | $( ".selector" ).slider({  
2 |   change: function( event, ui ) {}  
3 | });
```

Bind an event listener to the slidechange event:

```
1 | $( ".selector" ).on( "slidechange", func
```



## create( event, ui )

Type: `slidecreate`

Triggered when the slider is created.

**event**

Type: `Event`

**ui**

Type: `Object`

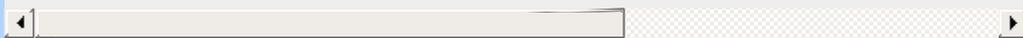
### Code examples:

Initialize the slider with the create callback specified:

```
1 | $( ".selector" ).slider({
2 |   create: function( event, ui ) {}
3 | });
```

Bind an event listener to the slidecreate event:

```
1 | $( ".selector" ).on( "slidecreate", function( event,
```



## slide( event, ui )

Type: `slide`

Triggered on every mouse move during slide. The value provided in the event as `ui.value` represents the value that the handle will have as a result of the current movement. Canceling the event will prevent the handle from moving and the handle will continue to have its previous value.

**event**

Type: `Event`

## ui

Type: [Object](#)

### handle

Type: [jQuery](#)

The jQuery object representing the handle being moved.

### value

Type: [Number](#)

The value that the handle will move to if the event is not canceled.

### values

Type: [Array](#)

An array of the current values of a multi-handled slider.

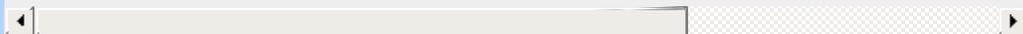
## Code examples:

Initialize the slider with the slide callback specified:

```
1 | $( ".selector" ).slider({  
2 |   slide: function( event, ui ) {}  
3 | });
```

Bind an event listener to the slide event:

```
1 | $( ".selector" ).on( "slide", function( event, ui ) {
```



## start( event, ui )

Type: [slidestart](#)

Triggered when the user starts sliding.

### event

Type: [Event](#)

**ui**

Type: Object

**handle**

Type: jQuery

The jQuery object representing the handle being moved.

**value**

Type: Number

The current value of the slider.

### Code examples:

Initialize the slider with the start callback specified:

```
1 | $( ".selector" ).slider({
2 |   start: function( event, ui ) {}
3 | });
```

Bind an event listener to the slidestart event:

```
1 | $( ".selector" ).on( "slidestart", function( event, ui
```



**stop( event, ui )**

Type: slidestop

Triggered after the user slides a handle.

**event**

Type: Event

**ui**

Type: Object

**handle**

Type: [jQuery](#)

The jQuery object representing the handle that was moved.

### value

Type: [Number](#)

The current value of the slider.

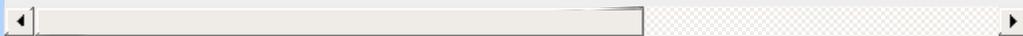
### Code examples:

Initialize the slider with the stop callback specified:

```
1 | $( ".selector" ).slider({  
2 |   stop: function( event, ui ) {}  
3 | });
```

Bind an event listener to the slidestop event:

```
1 | $( ".selector" ).on( "slidestop", function( event, ui
```



## Example:

*A simple jQuery UI Slider.*

```
1  <!doctype html>
2  <html lang="en">
3  <head>
4      <meta charset="utf-8">
5      <title>slider demo</title>
6      <link rel="stylesheet" href="http://code.jqu
7      <style>#slider { margin: 10px; } </sty.
8      <script src="http://code.jquery.com/jquery-:
9      <script src="http://code.jquery.com/ui/1.10
10 </head>
11 <body>
12
13 <div id="slider"></div>
14
15 <script>
16 $( "#slider" ).slider();
17 </script>
18
19 </body>
20 </html>
```

Demo

**A new version of this book is available!**

---

# Sortable Widget

Categories: [Interactions](#)

---

**Sortable Widget**

version added: 1.0

**Description:** *Reorder elements in a list or grid using the mouse.*

## QuickNav

## Examples

### Options

[appendTo](#)  
[axis](#)  
[cancel](#)  
[connectWith](#)  
[containment](#)  
[cursor](#)  
[cursorAt](#)  
[delay](#)  
[disabled](#)  
[distance](#)  
[dropOnEmpty](#)  
[forceHelperSize](#)  
[forcePlaceholderSize](#)  
[grid](#)  
[handle](#)  
[helper](#)  
[items](#)  
[opacity](#)  
[placeholder](#)  
[revert](#)  
[scroll](#)  
[scrollSensitivity](#)  
[scrollSpeed](#)  
[tolerance](#)  
[zIndex](#)

### Methods

[cancel](#)  
[destroy](#)  
[disable](#)  
[enable](#)  
[option](#)  
[refresh](#)  
[refreshPositions](#)  
[serialize](#)  
[toArray](#)  
[widget](#)

### Events

[create](#)  
[start](#)  
[sort](#)  
[change](#)  
[beforeStop](#)  
[stop](#)  
[update](#)  
[receive](#)  
[remove](#)  
[over](#)  
[out](#)  
[activate](#)  
[deactivate](#)

The jQuery UI Sortable plugin makes selected elements sortable by dragging with the mouse.

*Note: In order to sort table rows, the `tbody` must be made sortable, not the `table`.*

## Dependencies

UI Core

Widget Factory

Mouse Interaction

# Options

## appendTo

Default: **Type:** [jQuery](#) or [Element](#) or [Selector](#) or [String](#)  
"parent"

Defines where the helper that moves with the mouse is being appended to during the drag (for example, to resolve overlap/zIndex issues).

### Multiple types supported:

**jQuery:** A jQuery object containing the element to append the helper to.

**Element:** The element to append the helper to.

**Selector:** A selector specifying which element to append the helper to.

**String:** The string "parent" will cause the helper to be a sibling of the sortable item.

### Code examples:

Initialize the sortable with the appendTo option specified:

```
1 | $( ".selector" ).sortable({ appendTo: de
```

Get or set the appendTo option, after initialization:

```
1 | // getter
2 | var appendTo = $( ".selector" ).sortable
3 |
4 | // setter
5 | $( ".selector" ).sortable( "option", "ap
```

## axis

Type: [String](#)

Default: `false`

If defined, the items can be dragged only horizontally or vertically. Possible values: `"x"`, `"y"`.

### Code examples:

Initialize the sortable with the axis option specified:

```
1 | $( ".selector" ).sortable({ axis: "x" });
```

Get or set the axis option, after initialization:

```
1 | // getter
2 | var axis = $( ".selector" ).sortable( "option"
3 |
4 | // setter
5 | $( ".selector" ).sortable( "option", "axis",
```

## cancel

Type: [Selector](#)

Default: `":input,button"`

Prevents sorting if you start on elements matching the selector.

### Code examples:

Initialize the sortable with the cancel option specified:

```
1 | $( ".selector" ).sortable({ cancel: "a,button"
```

Get or set the cancel option, after initialization:

```
1 // getter
2 var cancel = $( ".selector" ).sortable( "option"
3
4 // setter
5 $( ".selector" ).sortable( "option", "cancel"
```

## connectWith

Type: [Selector](#)

Default: `false`

A selector of other sortable elements that the items from this list should be connected to. This is a one-way relationship, if you want the items to be connected in both directions, the `connectWith` option must be set on both sortable elements.

### Code examples:

Initialize the sortable with the connectWith option specified:

```
1 $( ".selector" ).sortable({ connectWith: "#shopping-c
```

Get or set the connectWith option, after initialization:

```
1 // getter
2 var connectWith = $( ".selector" ).sortable(
3
4 // setter
5 $( ".selector" ).sortable( "option", "connectWith"
```

**containment** Type: [Element](#) or [Selector](#) or [String](#)

Default: `false`

Defines a bounding box that the sortable items are constrained to while dragging.

Note: The element specified for containment must have a calculated width and height (though it need not be explicit). For example, if you have `float: left` sortable children and specify `containment: "parent"` be sure to have `float: left` on the sortable/parent container as well or it will have `height: 0`, causing undefined behavior.

### Multiple types supported:

**Element:** An element to use as the container.

**Selector:** A selector specifying an element to use as the container.

**String:** A string identifying an element to use as the container. Possible values: `"parent"`, `"document"`, `"window"`.

### Code examples:

Initialize the sortable with the containment option specified:

```
1 | $( ".selector" ).sortable({ containment: "parent"
```

Get or set the containment option, after initialization:

```
1 | // getter
2 | var containment = $( ".selector" ).sortable(
3 |
4 | // setter
5 | $( ".selector" ).sortable( "option", "containment"
```

**cursor**

**Type:** String

Default: "auto"

Defines the cursor that is being shown while sorting.

### Code examples:

Initialize the sortable with the cursor option specified:

```
1 | $( ".selector" ).sortable({ cursor: "move" });
```

Get or set the cursor option, after initialization:

```
1 | // getter
2 | var cursor = $( ".selector" ).sortable( "option"
3 |
4 | // setter
5 | $( ".selector" ).sortable( "option", "cursor"
```

## cursorAt

Type: [Object](#)

Default: false

Moves the sorting element or helper so the cursor always appears to drag from the same position. Coordinates can be given as a hash using a combination of one or two keys: { top, left, right, bottom }.

### Code examples:

Initialize the sortable with the cursorAt option specified:

```
1 | $( ".selector" ).sortable({ cursorAt: { left:
```

Get or set the cursorAt option, after initialization:

```
1 | // getter
```

```
2 var cursorAt = $( ".selector" ).sortable( "option"  
3  
4 // setter  
5 $( ".selector" ).sortable( "option", "cursorAt"
```

## delay

Type: [Integer](#)

Default: `0`

Time in milliseconds to define when the sorting should start. Adding a delay helps preventing unwanted drags when clicking on an element.

### Code examples:

Initialize the sortable with the delay option specified:

```
1 | $( ".selector" ).sortable({ delay: 150 });
```

Get or set the delay option, after initialization:

```
1 // getter  
2 var delay = $( ".selector" ).sortable( "option"  
3  
4 // setter  
5 $( ".selector" ).sortable( "option", "delay",
```

## disabled

Type: [Boolean](#)

Default: `false`

Disables the sortable if set to `true`.

### Code examples:

Initialize the sortable with the disabled option specified:

```
1 | $( ".selector" ).sortable({ disabled: true });
```

Get or set the disabled option, after initialization:

```
1 | // getter
2 | var disabled = $( ".selector" ).sortable( "option"
3 |
4 | // setter
5 | $( ".selector" ).sortable( "option", "disabled"
```

## distance

Type: [Number](#)

Default: [1](#)

Tolerance, in pixels, for when sorting should start. If specified, sorting will not start until after mouse is dragged beyond distance. Can be used to allow for clicks on elements within a handle.

### Code examples:

Initialize the sortable with the distance option specified:

```
1 | $( ".selector" ).sortable({ distance: 5 });
```

Get or set the distance option, after initialization:

```
1 | // getter
2 | var distance = $( ".selector" ).sortable( "option"
3 |
4 | // setter
5 | $( ".selector" ).sortable( "option", "distance"
```

## dropOnEmpty

Type: [Boolean](#)

Default: `true`

If `false`, items from this sortable can't be dropped on an empty connect sortable (see the [connectWith](#) option).

### Code examples:

Initialize the sortable with the `dropOnEmpty` option specified:

```
1 | $( ".selector" ).sortable({ dropOnEmpty: false
```

Get or set the `dropOnEmpty` option, after initialization:

```
1 | // getter
2 | var dropOnEmpty = $( ".selector" ).sortable(
3 |
4 | // setter
5 | $( ".selector" ).sortable( "option", "dropOnEmpty"
```

## forceHelperSize

Type: [Boolean](#)

Default: `false`

If `true`, forces the helper to have a size.

### Code examples:

Initialize the sortable with the `forceHelperSize` option specified:

```
1 | $( ".selector" ).sortable({ forceHelperSize:
```

Get or set the forceHelperSize option, after initialization:

```
1 // getter
2 var forceHelperSize = $( ".selector" ).sortable(
3
4 // setter
5 $( ".selector" ).sortable( "option", "forceHelperSize"
```

**forcePlaceholderSize**

Type: [Boolean](#)

Default: `false`

If true, forces the placeholder to have a size.

**Code examples:**

Initialize the sortable with the forcePlaceholderSize option specified:

```
1 $( ".selector" ).sortable({ forcePlaceholderSize:
```

Get or set the forcePlaceholderSize option, after initialization:

```
1 // getter
2 var forcePlaceholderSize = $( ".selector" ).sortable(
3
4 // setter
5 $( ".selector" ).sortable( "option", "forcePlaceholder"
```

**grid**

Type: [Array](#)

Default: `false`

Snaps the sorting element or helper to a grid, every x and y pixels. Array values: `[ x, y ]`.

### Code examples:

Initialize the sortable with the grid option specified:

```
1 | $( ".selector" ).sortable({ grid: [ 20, 10 ] });
```

Get or set the grid option, after initialization:

```
1 | // getter
2 | var grid = $( ".selector" ).sortable( "option"
3 |
4 | // setter
5 | $( ".selector" ).sortable( "option", "grid", [
```

## handle

Type: [Selector](#) or [Element](#)

Default: `false`

Restricts sort start click to the specified element.

### Code examples:

Initialize the sortable with the handle option specified:

```
1 | $( ".selector" ).sortable({ handle: ".handle"
```

Get or set the handle option, after initialization:

```
1 | // getter
2 | var handle = $( ".selector" ).sortable( "option"
3 |
```

```
4 // setter
5 $( ".selector" ).sortable( "option", "handle"
```

## helper

Type: [String](#) or [Function\(\)](#)

Default: `"original"`

Allows for a helper element to be used for dragging display.

### Multiple types supported:

**String:** If set to `"clone"`, then the element will be cloned and the clone will be dragged.

**Function:** A function that will return a `DOMElement` to use while dragging. The function receives the event and the element being sorted.

### Code examples:

Initialize the sortable with the helper option specified:

```
1 | $( ".selector" ).sortable({ helper: "clone" });
```

Get or set the helper option, after initialization:

```
1 // getter
2 var helper = $( ".selector" ).sortable( "option"
3
4 // setter
5 $( ".selector" ).sortable( "option", "helper"
```

## items

Type: [Selector](#)

Default: `"> *"`

Specifies which items inside the element should be sortable.

### Code examples:

Initialize the sortable with the items option specified:

```
1 | $( ".selector" ).sortable({ items: "> li" });
```

Get or set the items option, after initialization:

```
1 | // getter
2 | var items = $( ".selector" ).sortable( "option"
3 |
4 | // setter
5 | $( ".selector" ).sortable( "option", "items",
```

---

## opacity

Type: Number

Default: `false`

Defines the opacity of the helper while sorting. From `0.01` to `1`.

### Code examples:

Initialize the sortable with the opacity option specified:

```
1 | $( ".selector" ).sortable({ opacity: 0.5 });
```

Get or set the opacity option, after initialization:

```
1 | // getter
2 | var opacity = $( ".selector" ).sortable( "option"
3 |
```

```
4 // setter
5 $( ".selector" ).sortable( "option", "opacity"
```

## placeholder

Type: [String](#)

Default: `false`

A class name that gets applied to the otherwise white space.

### Code examples:

Initialize the sortable with the placeholder option specified:

```
1 | $( ".selector" ).sortable({ placeholder: "sortable-pi
```

Get or set the placeholder option, after initialization:

```
1 // getter
2 var placeholder = $( ".selector" ).sortable(
3
4 // setter
5 $( ".selector" ).sortable( "option", "placeholder"
```

## revert

Type: [Boolean](#) or [Number](#)

Default: `false`

Whether the sortable items should revert to their new positions using a smooth animation.

### Multiple types supported:

**Boolean:** When set to `true`, the items will animate with the default duration.

**Number:** The duration for the animation, in

milliseconds.

### Code examples:

Initialize the sortable with the revert option specified:

```
1 | $( ".selector" ).sortable({ revert: true });
```

Get or set the revert option, after initialization:

```
1 | // getter
2 | var revert = $( ".selector" ).sortable( "option"
3 |
4 | // setter
5 | $( ".selector" ).sortable( "option", "revert"
```

## scroll

Type: **Boolean**

Default: **true**

If set to true, the page scrolls when coming to an edge.

### Code examples:

Initialize the sortable with the scroll option specified:

```
1 | $( ".selector" ).sortable({ scroll: false });
```

Get or set the scroll option, after initialization:

```
1 | // getter
2 | var scroll = $( ".selector" ).sortable( "option"
3 |
4 | // setter
```

```
5 | $( ".selector" ).sortable( "option", "scroll"
```

## scrollSensitivity

Type: [Number](#)

Default: `20`

Defines how near the mouse must be to an edge to start scrolling.

### Code examples:

Initialize the sortable with the scrollSensitivity option specified:

```
1 | $( ".selector" ).sortable({ scrollSensitivity:
```

Get or set the scrollSensitivity option, after initialization:

```
1 | // getter
2 | var scrollSensitivity = $( ".selector" ).sortable(
3 |
4 | // setter
5 | $( ".selector" ).sortable( "option", "scrollSensitivity"
```

## scrollSpeed

Type: [Number](#)

Default: `20`

The speed at which the window should scroll once the mouse pointer gets within the `scrollSensitivity` distance.

### Code examples:

Initialize the sortable with the scrollSpeed option specified:

```
1 | $( ".selector" ).sortable({ scrollSpeed: 40 });
```

Get or set the scrollSpeed option, after initialization:

```
1 | // getter
2 | var scrollSpeed = $( ".selector" ).sortable(
3 |
4 | // setter
5 | $( ".selector" ).sortable( "option", "scrollSpeed"
```

## tolerance

Type: [String](#)

Default: `"intersect"`

Specifies which mode to use for testing whether the item being moved is hovering over another item. Possible values:

`"intersect"`: The item overlaps the other item by at least 50%.

`"pointer"`: The mouse pointer overlaps the other item.

### Code examples:

Initialize the sortable with the tolerance option specified:

```
1 | $( ".selector" ).sortable({ tolerance: "pointer"
```

Get or set the tolerance option, after initialization:

```
1 | // getter
2 | var tolerance = $( ".selector" ).sortable( "option"
3 |
4 | // setter
```

```
5 | $( ".selector" ).sortable( "option", "tolerance"
```

## zIndex

Type: Integer

Default: `1000`

Z-index for element/helper while being sorted.

### Code examples:

Initialize the sortable with the zIndex option specified:

```
1 | $( ".selector" ).sortable({ zIndex: 9999 });
```

Get or set the zIndex option, after initialization:

```
1 | // getter
2 | var zIndex = $( ".selector" ).sortable( "option"
3 |
4 | // setter
5 | $( ".selector" ).sortable( "option", "zIndex"
```

## Methods

### cancel()

Cancels a change in the current sortable and reverts it to the state prior to when the current sort was started. Useful in the stop and receive callback functions.

This method does not accept any arguments.

#### Code examples:

Invoke the cancel method:

```
1 | $( ".selector" ).sortable( "cancel" );
```

### destroy()

Removes the sortable functionality completely. This will return the element back to its pre-init state.

This method does not accept any arguments.

#### Code examples:

Invoke the destroy method:

```
1 | $( ".selector" ).sortable( "destroy" );
```

### disable()

Disables the sortable.

This method does not accept any arguments.

### Code examples:

Invoke the disable method:

```
1 | $( ".selector" ).sortable( "disable" );
```

## enable()

Enables the sortable.

This method does not accept any arguments.

### Code examples:

Invoke the enable method:

```
1 | $( ".selector" ).sortable( "enable" );
```

## option( optionName )

Returns: *Object*

Gets the value currently associated with the specified `optionName`.

**optionName**

Type: *String*

The name of the option to get.

### Code examples:

Invoke the method:

```
1 | var isDisabled = $( ".selector" ).sortable( "option"
```

## option()

Returns: [PlainObject](#)

Gets an object containing key/value pairs representing the current sortable options hash.

This method does not accept any arguments.

### Code examples:

Invoke the method:

```
1 | var options = $( ".selector" ).sortable( "option"
```

## option( optionName, value )

Sets the value of the sortable option associated with the specified `optionName`.

### optionName

Type: [String](#)

The name of the option to set.

### value

Type: [Object](#)

A value to set for the option.

### Code examples:

Invoke the method:

```
1 | $( ".selector" ).sortable( "option", "disabled"
```

## option( options )

Sets one or more options for the sortable.

### options

Type: [Object](#)

A map of option-value pairs to set.

### Code examples:

Invoke the method:

```
1 | $( ".selector" ).sortable( "option", { disabled:
```

## refresh()

Refresh the sortable items. Triggers the reloading of all sortable items, causing new items to be recognized.

This method does not accept any arguments.

### Code examples:

Invoke the refresh method:

```
1 | $( ".selector" ).sortable( "refresh" );
```

## refreshPositions()

Refresh the cached positions of the sortable items. Calling this method refreshes the cached item positions of all sortables.

This method does not accept any arguments.

## Code examples:

Invoke the refreshPositions method:

```
1 | $( ".selector" ).sortable( "refreshPositions"
```

## serialize( options )

Returns: *String*

Serializes the sortable's item `ids` into a form/ajax submittable string. Calling this method produces a hash that can be appended to any url to easily submit a new item order back to the server.

It works by default by looking at the `id` of each item in the format `"setname_number"`, and it spits out a hash like

```
"setname[]=number&setname[]=number".
```

*Note: If serialize returns an empty string, make sure the `id` attributes include an underscore. They must be in the form:*

`"set_number"` For example, a 3 element list with `id` attributes

```
"foo_1", "foo_5", "foo_2" will serialize to
```

```
"foo[]=1&foo[]=5&foo[]=2". You can use an underscore, equal sign or hyphen to separate the set and number. For
```

```
example "foo=1", "foo-1", and "foo_1" all serialize to
```

```
"foo[]=1".
```

### options

Type: *Object*

Options to customize the serialization.

**key** (default: `the part of the attribute in front of the separator`)

Type: *String*

Replaces `part1[]` with the specified value.

**attribute** (default: `"id"`)

Type: *String*

The name of the attribute to use for the values.

**expression** (default: `/(.+)[-=_](.+)/`)

Type: [RegExp](#)

A regular expression used to split the attribute value into key and value parts.

### Code examples:

Invoke the `serialize` method:

```
1 | var sorted = $( ".selector" ).sortable( "serialize"
```

## `toArray( options )`

Returns: [Array](#)

Serializes the sortable's item id's into an array of string.

### **options**

Type: [Object](#)

Options to customize the serialization.

**attribute** (default: `"id"`)

Type: [String](#)

The name of the attribute to use for the values.

### Code examples:

Invoke the `toArray` method:

```
1 | var sortedIDs = $( ".selector" ).sortable( "toArray"
```

## `widget()`

Returns: [jQuery](#)

Returns a [jQuery](#) object containing the sortable element.

This method does not accept any arguments.

**Code examples:**

Invoke the widget method:

```
1 | var widget = $( ".selector" ).sortable( "widget"
```

## Events

**activate( event, ui )**

Type: `sortactivate`

This event is triggered when using connected lists, every connected list on drag start receives it.

**event**

Type: [Event](#)

**ui**

Type: [Object](#)

**helper**

Type: [jQuery](#)

The jQuery object representing the helper being sorted

**item**

Type: [jQuery](#)

The jQuery object representing the current dragged element

**offset**

Type: [Object](#)

The current absolute position of the helper represented as `{ top, left }`

**position**

Type: [Object](#)

The current position of the helper represented as `{ top, left }`

**originalPosition**

Type: [Object](#)

The original position of the element represented as `{ top, left }`

**sender**Type: [jQuery](#)

The sortable that the item comes from if moving from one sortable to another

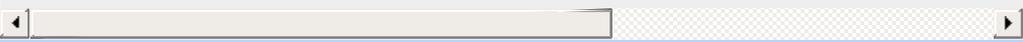
**Code examples:**

Initialize the sortable with the activate callback specified:

```
1 | $( ".selector" ).sortable({  
2 |   activate: function( event, ui ) {}  
3 | });
```

Bind an event listener to the sortactivate event:

```
1 | $( ".selector" ).on( "sortactivate", function
```



---

**beforeStop( event, ui )**    Type: `sortbeforestop`

This event is triggered when sorting stops, but when the placeholder/helper is still available.

**event**Type: [Event](#)**ui**Type: [Object](#)**helper**Type: [jQuery](#)

The jQuery object representing the helper being sorted

**item**Type: [jQuery](#)

The jQuery object representing the current dragged element

### **offset**

Type: Object

The current absolute position of the helper represented as `{ top, left }`

### **position**

Type: Object

The current position of the helper represented as `{ top, left }`

### **originalPosition**

Type: Object

The original position of the element represented as `{ top, left }`

### **sender**

Type: jQuery

The sortable that the item comes from if moving from one sortable to another

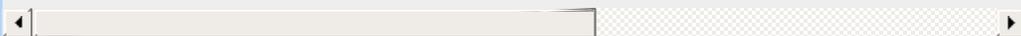
## **Code examples:**

Initialize the sortable with the `beforeStop` callback specified:

```
1 | $( ".selector" ).sortable({  
2 |   beforeStop: function( event, ui ) {}  
3 | });
```

Bind an event listener to the `sortbeforestop` event:

```
1 | $( ".selector" ).on( "sortbeforestop", function
```



## change( event, ui )

Type: `sortchange`

This event is triggered during sorting, but only when the DOM position has changed.

### event

Type: `Event`

### ui

Type: `Object`

#### helper

Type: `jQuery`

The jQuery object representing the helper being sorted

#### item

Type: `jQuery`

The jQuery object representing the current dragged element

#### offset

Type: `Object`

The current absolute position of the helper represented as `{ top, left }`

#### position

Type: `Object`

The current position of the helper represented as `{ top, left }`

#### originalPosition

Type: `Object`

The original position of the element represented as `{ top, left }`

#### sender

Type: `jQuery`

The sortable that the item comes from if moving from one sortable to another

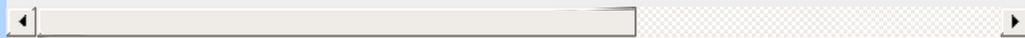
### Code examples:

Initialize the sortable with the change callback specified:

```
1 | $( ".selector" ).sortable({  
2 |   change: function( event, ui ) {}  
3 | });
```

Bind an event listener to the sortchange event:

```
1 | $( ".selector" ).on( "sortchange", function( event, ui
```



### create( event, ui )

Type: `sortcreate`

Triggered when the sortable is created.

**event**

Type: [Event](#)

**ui**

Type: [Object](#)

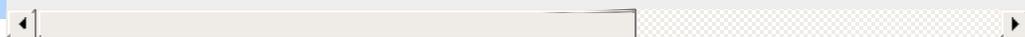
### Code examples:

Initialize the sortable with the create callback specified:

```
1 | $( ".selector" ).sortable({  
2 |   create: function( event, ui ) {}  
3 | });
```

Bind an event listener to the sortcreate event:

```
1 | $( ".selector" ).on( "sortcreate", function( event, ui
```



**deactivate( event, ui )**      *Type:* `sortdeactivate`

This event is triggered when sorting was stopped, is propagated to all possible connected lists.

**event**  
Type: [Event](#)

**ui**  
Type: [Object](#)

**helper**  
Type: [jQuery](#)  
The jQuery object representing the helper being sorted

**item**  
Type: [jQuery](#)  
The jQuery object representing the current dragged element

**offset**  
Type: [Object](#)  
The current absolute position of the helper represented as `{ top, left }`

**position**  
Type: [Object](#)  
The current position of the helper represented as `{ top, left }`

**originalPosition**  
Type: [Object](#)  
The original position of the element represented as `{ top, left }`

**sender**  
Type: [jQuery](#)  
The sortable that the item comes from if moving

from one sortable to another

### Code examples:

Initialize the sortable with the deactivate callback specified:

```
1 | $( ".selector" ).sortable({  
2 |   deactivate: function( event, ui ) {}  
3 | });
```

Bind an event listener to the sortdeactivate event:

```
1 | $( ".selector" ).on( "sortdeactivate", function
```



**out( event, ui )**

Type: `sortout`

This event is triggered when a sortable item is moved away from a connected list.

*Note: This event is also triggered when a sortable item is dropped.*

**event**

Type: [Event](#)

**ui**

Type: [Object](#)

**helper**

Type: [jQuery](#)

The jQuery object representing the helper being sorted

**item**

Type: [jQuery](#)

The jQuery object representing the current dragged element

### **offset**

Type: [Object](#)

The current absolute position of the helper represented as `{ top, left }`

### **position**

Type: [Object](#)

The current position of the helper represented as `{ top, left }`

### **originalPosition**

Type: [Object](#)

The original position of the element represented as `{ top, left }`

### **sender**

Type: [jQuery](#)

The sortable that the item comes from if moving from one sortable to another

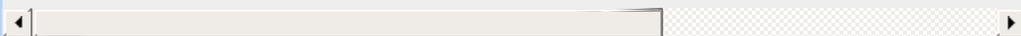
## **Code examples:**

Initialize the sortable with the out callback specified:

```
1 | $( ".selector" ).sortable({  
2 |   out: function( event, ui ) {}  
3 | });
```

Bind an event listener to the sortout event:

```
1 | $( ".selector" ).on( "sortout", function( event, ui )
```



## over( event, ui )

Type: `sortover`

This event is triggered when a sortable item is moved into a connected list.

### event

Type: `Event`

### ui

Type: `Object`

#### helper

Type: `jQuery`

The jQuery object representing the helper being sorted

#### item

Type: `jQuery`

The jQuery object representing the current dragged element

#### offset

Type: `Object`

The current absolute position of the helper represented as `{ top, left }`

#### position

Type: `Object`

The current position of the helper represented as `{ top, left }`

#### originalPosition

Type: `Object`

The original position of the element represented as `{ top, left }`

#### sender

Type: `jQuery`

The sortable that the item comes from if moving from one sortable to another

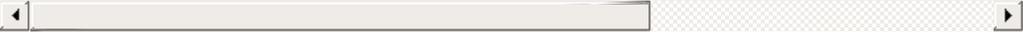
## Code examples:

Initialize the sortable with the over callback specified:

```
1 | $( ".selector" ).sortable({  
2 |   over: function( event, ui ) {}  
3 | });
```

Bind an event listener to the sortover event:

```
1 | $( ".selector" ).on( "sortover", function( event, ui
```



---

## receive( event, ui )

Type: `sortreceive`

This event is triggered when a connected sortable list has received an item from another list.

### event

Type: `Event`

### ui

Type: `Object`

#### helper

Type: `jQuery`

The jQuery object representing the helper being sorted

#### item

Type: `jQuery`

The jQuery object representing the current dragged element

#### offset

Type: `Object`

The current absolute position of the helper represented as `{ top, left }`

### position

Type: Object

The current position of the helper represented as {  
top, left }

### originalPosition

Type: Object

The original position of the element represented as  
{ top, left }

### sender

Type: jQuery

The sortable that the item comes from if moving  
from one sortable to another

## Code examples:

Initialize the sortable with the receive callback specified:

```
1 | $( ".selector" ).sortable({  
2 |   receive: function( event, ui ) {}  
3 | });
```

Bind an event listener to the sortreceive event:

```
1 | $( ".selector" ).on( "sortreceive", function( event,
```

---

## remove( event, ui )

Type: sortremove

This event is triggered when a sortable item has been  
dragged out from the list and into another.

### event

Type: Event

## ui

Type: [Object](#)

### helper

Type: [jQuery](#)

The jQuery object representing the helper being sorted

### item

Type: [jQuery](#)

The jQuery object representing the current dragged element

### offset

Type: [Object](#)

The current absolute position of the helper represented as `{ top, left }`

### position

Type: [Object](#)

The current position of the helper represented as `{ top, left }`

### originalPosition

Type: [Object](#)

The original position of the element represented as `{ top, left }`

### sender

Type: [jQuery](#)

The sortable that the item comes from if moving from one sortable to another

## Code examples:

Initialize the sortable with the remove callback specified:

```
1 | $( ".selector" ).sortable({  
2 |   remove: function( event, ui ) {}  
3 | });
```

Bind an event listener to the sortremove event:

```
1 | $( ".selector" ).on( "sortremove", function( event, ui )
```

## sort( event, ui )

Type: `sort`

This event is triggered during sorting.

### event

Type: `Event`

### ui

Type: `Object`

#### helper

Type: `jQuery`

The jQuery object representing the helper being sorted

#### item

Type: `jQuery`

The jQuery object representing the current dragged element

#### offset

Type: `Object`

The current absolute position of the helper represented as `{ top, left }`

#### position

Type: `Object`

The current position of the helper represented as `{ top, left }`

#### originalPosition

Type: `Object`

The original position of the element represented as

```
{ top, left }
```

### sender

Type: [jQuery](#)

The sortable that the item comes from if moving from one sortable to another

## Code examples:

Initialize the sortable with the sort callback specified:

```
1 | $( ".selector" ).sortable({  
2 |   sort: function( event, ui ) {}  
3 | });
```

Bind an event listener to the sort event:

```
1 | $( ".selector" ).on( "sort", function( event, ui ) {
```



---

## start( event, ui )

Type: `sortstart`

This event is triggered when sorting starts.

### event

Type: [Event](#)

### ui

Type: [Object](#)

#### helper

Type: [jQuery](#)

The jQuery object representing the helper being sorted

#### item

Type: [jQuery](#)

The jQuery object representing the current dragged element

### **offset**

Type: [Object](#)

The current absolute position of the helper represented as `{ top, left }`

### **position**

Type: [Object](#)

The current position of the helper represented as `{ top, left }`

### **originalPosition**

Type: [Object](#)

The original position of the element represented as `{ top, left }`

### **sender**

Type: [jQuery](#)

The sortable that the item comes from if moving from one sortable to another

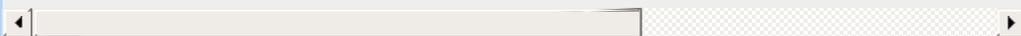
## **Code examples:**

Initialize the sortable with the start callback specified:

```
1 | $( ".selector" ).sortable({
2 |   start: function( event, ui ) {}
3 | });
```

Bind an event listener to the sortstart event:

```
1 | $( ".selector" ).on( "sortstart", function( event, ui
```



## stop( event, ui )

Type: `sortstop`

This event is triggered when sorting has stopped.

### event

Type: `Event`

### ui

Type: `Object`

#### helper

Type: `jQuery`

The jQuery object representing the helper being sorted

#### item

Type: `jQuery`

The jQuery object representing the current dragged element

#### offset

Type: `Object`

The current absolute position of the helper represented as `{ top, left }`

#### position

Type: `Object`

The current position of the helper represented as `{ top, left }`

#### originalPosition

Type: `Object`

The original position of the element represented as `{ top, left }`

#### sender

Type: `jQuery`

The sortable that the item comes from if moving from one sortable to another

**Code examples:**

Initialize the sortable with the stop callback specified:

```
1 | $( ".selector" ).sortable({  
2 |   stop: function( event, ui ) {}  
3 | });
```

Bind an event listener to the sortstop event:

```
1 | $( ".selector" ).on( "sortstop", function( event, ui
```

---

**update( event, ui )**

*Type:* `sortupdate`

This event is triggered when the user stopped sorting and the DOM position has changed.

**event**

Type: [Event](#)

**ui**

Type: [Object](#)

**helper**

Type: [jQuery](#)

The jQuery object representing the helper being sorted

**item**

Type: [jQuery](#)

The jQuery object representing the current dragged element

**offset**

Type: [Object](#)

The current absolute position of the helper represented as `{ top, left }`

### position

Type: [Object](#)

The current position of the helper represented as {  
top, left }

### originalPosition

Type: [Object](#)

The original position of the element represented as  
{ top, left }

### sender

Type: [jQuery](#)

The sortable that the item comes from if moving  
from one sortable to another

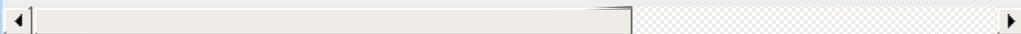
## Code examples:

Initialize the sortable with the update callback specified:

```
1 | $( ".selector" ).sortable({  
2 |   update: function( event, ui ) {}  
3 | });
```

Bind an event listener to the sortupdate event:

```
1 | $( ".selector" ).on( "sortupdate", function( event, ui )
```



## Example:

### *A simple jQuery UI Sortable.*

```
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>sortable demo</title>
6   <link rel="stylesheet" href="http://code.jq
7   <script src="http://code.jquery.com/jquery-:
8   <script src="http://code.jquery.com/ui/1.10
9 </head>
10 <body>
11
12 <ul id="sortable">
13   <li>Item 1</li>
14   <li>Item 2</li>
15   <li>Item 3</li>
16   <li>Item 4</li>
17   <li>Item 5</li>
18 </ul>
19
20 <script>$("#sortable").sortable();</script>
21
22 </body>
23 </html>
```

Demo

**A new version of this book is available!**

---

# Spinner Widget

Categories: [Widgets](#)

---

## Spinner Widget

version added: 1.9

**Description:** *Enhance a text input for entering numeric values, with up/down buttons and arrow key handling.*

## QuickNav

## Examples

### Options

[culture](#)  
[disabled](#)  
[icons](#)  
[incremental](#)  
[max](#)  
[min](#)  
[numberFormat](#)  
[page](#)  
[step](#)

### Methods

[destroy](#)  
[disable](#)  
[enable](#)  
[option](#)  
[pageDown](#)  
[pageUp](#)  
[stepDown](#)  
[stepUp](#)  
[value](#)  
[widget](#)

### Events

[create](#)  
[start](#)  
[spin](#)  
[stop](#)  
[change](#)

Spinner, or number stepper, widget is perfect for handling all kinds of numeric input. It allow users to type a value directly or modify an existing value by spinning with the keyboard, mouse or scrollwheel. When combined with [Globalize](#), you can even spin currencies and dates in a variety of locales.

Spinner wraps a text input, adds two buttons to increment and decrement the current value, along with handling key events for the same purpose. It delegates to [Globalize](#) for number formatting and parsing.

## Keyboard interaction

UP: Increment the value by one step.

DOWN: Decrement the value by one step.

PAGE UP: Increment the value by one page.

PAGE DOWN: Decrement the value by one page.

Focus stays in the text field, even after using the mouse to click one of the spin buttons.

## Dependencies

[UI Core](#)

[Widget Factory](#)

[Button](#)

[Globalize](#) (external, optional; for use with the [culture](#) and [numberFormat](#) options)

## Additional Notes:

This widget requires some functional CSS, otherwise it won't work. If you build a custom theme, use the widget's specific CSS file as a starting point.

This widget manipulates its element's value programmatically, therefore a native change may not be fired when the element's value changes.

## Options

### culture

Type: [String](#)

Default: `null`

Sets the culture to use for parsing and formatting the value. If `null`, the currently set culture in [Globalize](#) is used, see [Globalize docs](#) for available cultures. Only relevant if the [numberFormat](#) option is set. Requires [Globalize](#) to be included.

#### Code examples:

Initialize the spinner with the culture option specified:

```
1 | $( ".selector" ).spinner({ culture: "fr'
```

Get or set the culture option, after initialization:

```
1 | // getter
2 | var culture = $( ".selector" ).spinner(
3 |
4 | // setter
5 | $( ".selector" ).spinner( "option", "cu:
```

### disabled

Type: [Boolean](#)

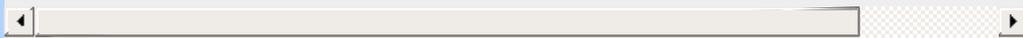
Default: `false`

Disables the spinner if set to `true`.

#### Code examples:

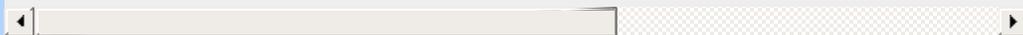
Initialize the spinner with the disabled option specified:

```
1 | $( ".selector" ).spinner({ disabled: true });
```



Get or set the disabled option, after initialization:

```
1 | // getter
2 | var disabled = $( ".selector" ).spinner( "option"
3 |
4 | // setter
5 | $( ".selector" ).spinner( "option", "disabled"
```



## icons

Type: [Object](#)

Default: `{ down: "ui-icon-triangle-1-s", up: "ui-icon-triangle-1-n" }`

Icons to use for buttons, matching an icon defined by the jQuery UI CSS Framework.

up (string, default: "ui-icon-triangle-1-n")

down (string, default: "ui-icon-triangle-1-s")

### Code examples:

Initialize the spinner with the icons option specified:

```
1 | $( ".selector" ).spinner({ icons: { down: "custom-dow
```



Get or set the icons option, after initialization:

```
1 | // getter
2 | var icons = $( ".selector" ).spinner( "option"
3 |
4 | // setter
```

```
5 $( ".selector" ).spinner( "option", "icons", { down:
```

## incremental

Default: `true` Type: [Boolean](#) or [Function](#)( [Integer](#) count )

Controls the number of steps taken when holding down a spin button.

### Multiple types supported:

**Boolean:** When set to `true`, the stepping delta will increase when spun incessantly. When set to `false`, all steps are equal (as defined by the `step` option).

**Function:** Receives one parameter: the number of spins that have occurred. Must return the number of steps that should occur for the current spin.

### Code examples:

Initialize the spinner with the incremental option specified:

```
1 | $( ".selector" ).spinner({ incremental: false
```

Get or set the incremental option, after initialization:

```
1 | // getter
2 | var incremental = $( ".selector" ).spinner( "option"
3 |
4 | // setter
5 | $( ".selector" ).spinner( "option", "incremental"
```

max

Type: [Number](#) or [String](#)

Default: `null`

The maximum allowed value. The element's `max` attribute is used if it exists and the option is not explicitly set. If `null`, there is no maximum enforced.

### Multiple types supported:

**Number:** The maximum value.

**String:** If [Globalize](#) is included, the `max` option can be passed as a string which will be parsed based on the `numberFormat` and `culture` options; otherwise it will fall back to the native `parseFloat()` method.

### Code examples:

Initialize the spinner with the max option specified:

```
1 | $( ".selector" ).spinner({ max: 50 });
```

Get or set the max option, after initialization:

```
1 | // getter
2 | var max = $( ".selector" ).spinner( "option",
3 |
4 | // setter
5 | $( ".selector" ).spinner( "option", "max", 50
```

## min

Type: [Number](#) or [String](#)

Default: `null`

The minimum allowed value. The element's `min` attribute is used if it exists and the option is not explicitly set. If `null`, there is no minimum enforced.

### Multiple types supported:

**Number:** The minimum value.

**String:** If [Globalize](#) is included, the `min` option can be

passed as a string which will be parsed based on the `numberFormat` and `culture` options; otherwise it will fall back to the native `parseFloat()` method.

### Code examples:

Initialize the spinner with the min option specified:

```
1 | $( ".selector" ).spinner({ min: 0 });
```

Get or set the min option, after initialization:

```
1 | // getter
2 | var min = $( ".selector" ).spinner( "option",
3 |
4 | // setter
5 | $( ".selector" ).spinner( "option", "min", 0 );
```

## numberFormat

Type: [String](#)

Default: `null`

Format of numbers passed to `globalize`, if available. Most common are `"n"` for a decimal number and `"c"` for a currency value. Also see the `culture` option.

### Code examples:

Initialize the spinner with the numberFormat option specified:

```
1 | $( ".selector" ).spinner({ numberFormat: "n" });
```

Get or set the numberFormat option, after initialization:

```
1 // getter
2 var numberFormat = $( ".selector" ).spinner(
3
4 // setter
5 $( ".selector" ).spinner( "option", "numberFormat"
```

**page**

**Type:** [Number](#)

**Default:** 10

The number of steps to take when paging via the [pageUp/pageDown](#) methods.

**Code examples:**

Initialize the spinner with the page option specified:

```
1 | $( ".selector" ).spinner({ page: 5 });
```

Get or set the page option, after initialization:

```
1 // getter
2 var page = $( ".selector" ).spinner( "option"
3
4 // setter
5 $( ".selector" ).spinner( "option", "page", 5
```

**step**

**Type:** [Number](#) or [String](#)

**Default:** 1

The size of the step to take when spinning via buttons or via the [stepUp\(\)/stepDown\(\)](#) methods. The element's `step` attribute is used if it exists and the option is not explicitly set.

## Multiple types supported:

**Number:** The size of the step.

**String:** If Globalize is included, the `step` option can be passed as a string which will be parsed based on the `numberFormat` and `culture` options, otherwise it will fall back to the native `parseFloat`.

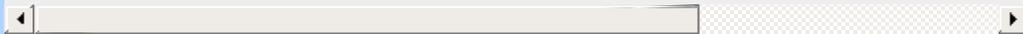
## Code examples:

Initialize the spinner with the step option specified:

```
1 | $( ".selector" ).spinner({ step: 2 });
```

Get or set the step option, after initialization:

```
1 | // getter
2 | var step = $( ".selector" ).spinner( "option"
3 |
4 | // setter
5 | $( ".selector" ).spinner( "option", "step", 2
```



## Methods

### destroy()

Removes the spinner functionality completely. This will return the element back to its pre-init state.

This method does not accept any arguments.

#### Code examples:

Invoke the destroy method:

```
1 | $( ".selector" ).spinner( "destroy" );
```

### disable()

Disables the spinner.

This method does not accept any arguments.

#### Code examples:

Invoke the disable method:

```
1 | $( ".selector" ).spinner( "disable" );
```

### enable()

Enables the spinner.

This method does not accept any arguments.

### Code examples:

Invoke the enable method:

```
1 | $( ".selector" ).spinner( "enable" );
```

## option( optionName )

Returns: *Object*

Gets the value currently associated with the specified `optionName`.

**optionName**

Type: *String*

The name of the option to get.

### Code examples:

Invoke the method:

```
1 | var isDisabled = $( ".selector" ).spinner( "option"
```



## option()

Returns: *PlainObject*

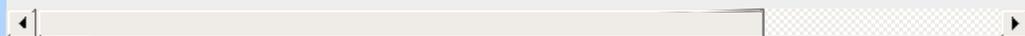
Gets an object containing key/value pairs representing the current spinner options hash.

This method does not accept any arguments.

### Code examples:

Invoke the method:

```
1 | var options = $( ".selector" ).spinner( "option"
```



## option( optionName, value )

Sets the value of the spinner option associated with the specified `optionName`.

**optionName**

Type: String

The name of the option to set.

**value**

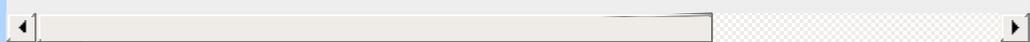
Type: Object

A value to set for the option.

**Code examples:**

Invoke the method:

```
1 | $( ".selector" ).spinner( "option", "disabled"
```



## option( options )

Sets one or more options for the spinner.

**options**

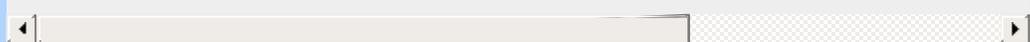
Type: Object

A map of option-value pairs to set.

**Code examples:**

Invoke the method:

```
1 | $( ".selector" ).spinner( "option", { disabled:
```



## pageDown( [pages ] )

Decrements the value by the specified number of pages, as defined by the `page` option. Without the parameter, a single page is decremented.

If the resulting value is above the max, below the min, or results in a step mismatch, the value will be adjusted to the closest valid value.

Invoking `pageDown()` will cause `start`, `spin`, and `stop` events to be triggered.

### pages

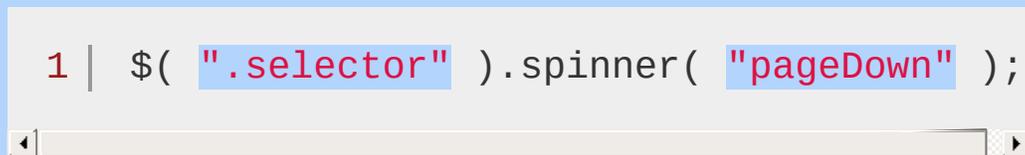
Type: `Number`

Number of pages to decrement, defaults to 1.

### Code examples:

Invoke the `pageDown` method:

```
1 | $( ".selector" ).spinner( "pageDown" );
```

A code editor window with a light gray background and a white text area. The text area contains the code: `1 | $( ".selector" ).spinner( "pageDown" );`. The code is highlighted with a light blue background. The editor has a scrollbar on the right side.

## pageUp( [pages ] )

Increments the value by the specified number of pages, as defined by the `page` option. Without the parameter, a single page is incremented.

If the resulting value is above the max, below the min, or results in a step mismatch, the value will be adjusted to the closest valid value.

Invoking `pageUp()` will cause `start`, `spin`, and `stop` events to be triggered.

### pages

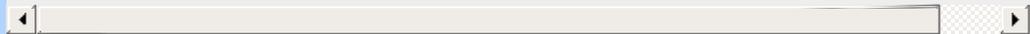
Type: [Number](#)

Number of pages to increment, defaults to 1.

### Code examples:

Invoke the pageUp method:

```
1 | $( ".selector" ).spinner( "pageUp", 10 );
```



## stepDown( [steps ] )

Decrements the value by the specified number of steps. Without the parameter, a single step is decremented.

If the resulting value is above the max, below the min, or results in a step mismatch, the value will be adjusted to the closest valid value.

Invoking `stepDown()` will cause `start`, `spin`, and `stop` events to be triggered.

### steps

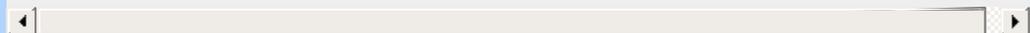
Type: [Number](#)

Number of steps to decrement, defaults to 1.

### Code examples:

Invoke the stepDown method:

```
1 | $( ".selector" ).spinner( "stepDown" );
```



## stepUp( [steps ] )

Increments the value by the specified number of steps.

Without the parameter, a single step is incremented.

If the resulting value is above the max, below the min, or results in a step mismatch, the value will be adjusted to the closest valid value.

Invoking `stepUp()` will cause `start`, `spin`, and `stop` events to be triggered.

### **steps**

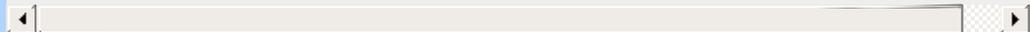
Type: Number

Number of steps to increment, defaults to 1.

### **Code examples:**

Invoke the `stepUp` method:

```
1 | $( ".selector" ).spinner( "stepUp", 5 );
```



## **value()**

Returns: Number

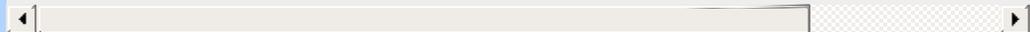
Gets the current value as a number. The value is parsed based on the `numberFormat` and `culture` options.

This method does not accept any arguments.

### **Code examples:**

Invoke the method:

```
1 | var value = $( ".selector" ).spinner( "value"
```



## **value( value )**

## value

Type: [Number](#) or [String](#)

The value to set. If passed as a string, the value is parsed based on the [numberFormat](#) and [culture](#) options.

### Code examples:

Invoke the method:

```
1 | $( ".selector" ).spinner( "value", 50 );
```

## widget()

Returns: [jQuery](#)

Returns a [jQuery](#) object containing the generated wrapper.

This method does not accept any arguments.

### Code examples:

Invoke the widget method:

```
1 | var widget = $( ".selector" ).spinner( "widget" );
```

## Events

### change( event, ui )

Type: `spinchange`

Triggered when the value of the spinner has changed and the input is no longer focused.

**event**

Type: [Event](#)

**ui**

Type: [Object](#)

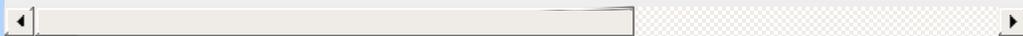
#### Code examples:

Initialize the spinner with the change callback specified:

```
1 | $( ".selector" ).spinner({  
2 |   change: function( event, ui ) {}  
3 | });
```

Bind an event listener to the spinchange event:

```
1 | $( ".selector" ).on( "spinchange", funct
```



### create( event, ui )

Type: `spincreate`

Triggered when the spinner is created.

**event**

Type: [Event](#)

**ui**

Type: [Object](#)

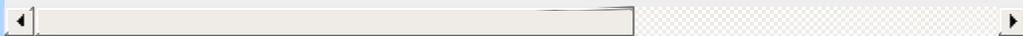
### Code examples:

Initialize the spinner with the create callback specified:

```
1 | $( ".selector" ).spinner({  
2 |   create: function( event, ui ) {}  
3 | });
```

Bind an event listener to the spincreate event:

```
1 | $( ".selector" ).on( "spincreate", function( event, ui )
```



**spin( event, ui )**

Type: [spin](#)

Triggered during increment/decrement (to determine direction of spin compare current value with `ui.value`). Can be canceled, preventing the value from being updated.

**event**

Type: [Event](#)

**ui**

Type: [Object](#)

**value**

Type: [Number](#)

The new value to be set, unless the event is cancelled.

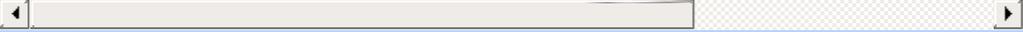
### Code examples:

Initialize the spinner with the spin callback specified:

```
1 | $( ".selector" ).spinner({
2 |   spin: function( event, ui ) {}
3 | });
```

Bind an event listener to the spin event:

```
1 | $( ".selector" ).on( "spin", function( event, ui ) {
```



**start( event, ui )**

Type: `spinstart`

Triggered before a spin. Can be canceled, preventing the spin from occurring.

**event**

Type: `Event`

**ui**

Type: `Object`

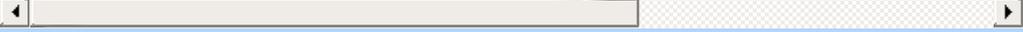
**Code examples:**

Initialize the spinner with the start callback specified:

```
1 | $( ".selector" ).spinner({
2 |   start: function( event, ui ) {}
3 | });
```

Bind an event listener to the spinstart event:

```
1 | $( ".selector" ).on( "spinstart", function( event, ui
```



## stop( event, ui )

Type: `spinstop`

Triggered after a spin.

**event**  
Type: `Event`

**ui**  
Type: `Object`

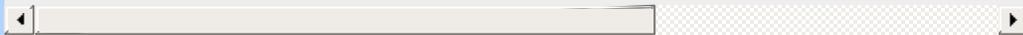
### Code examples:

Initialize the spinner with the stop callback specified:

```
1 | $( ".selector" ).spinner({  
2 |   stop: function( event, ui ) {}  
3 | });
```

Bind an event listener to the spinstop event:

```
1 | $( ".selector" ).on( "spinstop", function( event, ui
```



## Example:

### *Plain number spinner*

```
1  <!doctype html>
2  <html lang="en">
3  <head>
4      <meta charset="utf-8">
5      <title>spinner demo</title>
6      <link rel="stylesheet" href="http://code.jq
7      <script src="http://code.jquery.com/jquery-:
8      <script src="http://code.jquery.com/ui/1.10
9  </head>
10 <body>
11
12 <input id="spinner">
13
14 <script>
15 $( "#spinner" ).spinner();
16 </script>
17
18 </body>
19 </html>
```

Demo

**A new version of this book is available!**

---

# **.switchClass()**

Categories: [Effects](#) | [Effects Core](#)

```
.switchClass( removeClassName, addClassName [, duration ] [, easing ] [, complete ] ) Returns: jQuery
```

**Description:** *Adds and removes the specified class(es) to each of the set of matched elements while animating all style changes.*

```
.switchClass( removeClassName, addClassName [, duration ] [, easing ] [, complete ] )
```

#### **removeClassName**

Type: String

One or more class names (space separated) to be removed from the class attribute of each matched element.

#### **addClassName**

Type: String

One or more class names (space separated) to be added to the class attribute of each matched element.

**duration** (default: `400`)

Type: Number or String

A string or number determining how long the animation will run.

**easing** (default: `swing`)

Type: String

A string indicating which easing function to use for the transition.

#### **complete**

Type: Function()

A function to call once the animation is complete.

The `.switchClass()` method allows you to animate the transition of adding and removing classes at the same time.

Similar to native CSS transitions, jQuery UI's class animations provide a smooth transition from one state to another while allowing you to keep all the details about which styles to change in CSS and out of your JavaScript. All class animation methods, including

`.switchClass()`, support custom durations and easings, as well as providing a callback for when the animation completes.

Not all styles can be animated. For example, there is no way to animate a background image. Any styles that cannot be animated will be changed at the end of the animation.

This plugin extends jQuery's built-in `.switchClass()` method. If jQuery UI is not loaded, calling the `.switchClass()` method may not fail directly, as the method still exists. However, the expected behavior will not occur.

## Example:

*Adds the class "blue" and removes the class "big" from the matched elements.*

```
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>switchClass demo</title>
6   <link rel="stylesheet" href="http://code.jq
7   <style>
8   div {
9     width: 100px;
10    height: 100px;
11    background-color: #ccc;
12  }
13  .big {
14    width: 200px;
15    height: 200px;
16  }
17  .blue {
18    background-color: #00f;
19  }
20 </style>
21 <script src="http://code.jquery.com/jquery-:
22 <script src="http://code.jquery.com/ui/1.10
23 </head>
24 <body>
25
26 <div class="big"></div>
27
28 <script>
29 $( "div" ).click(function() {
30   $( this ).switchClass( "big", "blue", 1000,
```

```
31  });  
32  </script>  
33  
34  </body>  
35  </html>
```

Demo

POWERED BYHERONOTE

A new version of this book is available!

---

# :tabbable Selector

Categories: [Selectors](#) | [UI Core](#)

## tabbable selector

**Description:** *Selects elements which the user can focus via tabbing.*

### jQuery( ":tabbable" )

Some elements are natively tabbable, while others require explicitly setting a positive tab index. In all cases, the element must be visible in order to be tabbable.

Elements of the following type are tabbable if they do not have a negative tab index and are not disabled: `input`, `select`, `textarea`, `button`, and `object`. Anchors are focusable if they have an `href` or positive `tabindex` attribute. `area` elements are focusable if they are inside a named map, have an `href` attribute, and there is a visible image using the map. All other elements are tabbable based solely on their `tabindex` attribute and visibility.

*Note: Elements with a negative tab index are `:focusable`, but not `:tabbable`.*

## Example:

*Select tabbable elements and highlight them with a red border.*

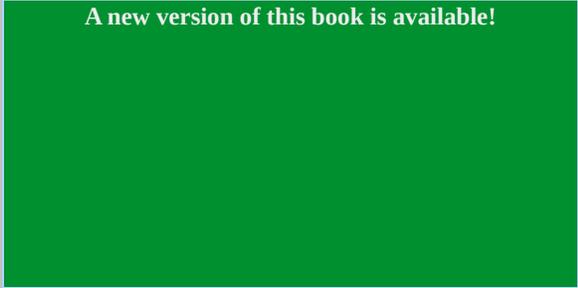
```
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>tabbable demo</title>
6   <link rel="stylesheet" href="http://code.jq
7   <style>
8   input {
9     border: 1px solid #000;
10  }
11  div {
12    padding: 5px;
13  }
14 </style>
15 <script src="http://code.jquery.com/jquery-:
16 <script src="http://code.jquery.com/ui/1.10
17 </head>
18 <body>
19
20 <div><input value="no tabindex"></div>
21 <div><input tabindex="5" value="positive tabi
22 <div><input tabindex="-1" value="negative tab.
23
24 <script>
25 $( ":tabbable" ).css( "border-color", "red" )
26 </script>
27
28 </body>
29 </html>
```



Demo

POWERED BYHERONOTE

A new version of this book is available!



---

# Tabs Widget

Categories: [Widgets](#)

---

**Tabs Widget**

version added: 1.0

**Description:** *A single content area with multiple panels, each associated with a header in a list.*

## QuickNav

## Examples

### Options

[active](#)  
[collapsible](#)  
[disabled](#)  
[event](#)  
[heightStyle](#)  
[hide](#)  
[show](#)

### Methods

[destroy](#)  
[disable](#)  
[enable](#)  
[load](#)  
[option](#)  
[refresh](#)  
[widget](#)

### Events

[activate](#)  
[beforeActivate](#)  
[beforeLoad](#)  
[create](#)  
[load](#)

Tabs are generally used to break content into multiple sections that can be swapped to save space, much like an accordion.

The content for each tab panel can be defined in-page or can be loaded via Ajax; both are handled automatically based on the `href` of the anchor associated with the tab. By default tabs are activated on click, but the events can be changed to hover via the `event` option.

## Keyboard interaction

When focus is on a tab, the following key commands are available:

UP/LEFT: Move focus to the previous tab. If on first tab, moves focus to last tab. Activate focused tab after a short delay.

DOWN/RIGHT: Move focus to the next tab. If on last tab, moves focus to first tab. Activate focused tab after a short delay.

HOME: Move focus to the first tab. Activate focused tab after a short delay.

END: Move focus to the last tab. Activate focused tab after a short delay.

SPACE: Activate panel associated with focused tab.

ENTER: Activate or toggle panel associated with focused tab.

ALT+PAGE UP: Move focus to the previous tab and immediately activate.

ALT+PAGE DOWN: Move focus to the next tab and immediately activate.

When focus is in a panel, the following key commands are available:

CTRL+UP: Move focus to associated tab.

ALT+PAGE UP: Move focus to the previous tab and immediately activate.

ALT+PAGE DOWN: Move focus to the next tab and immediately activate.

## Dependencies

[UI Core](#)

[Widget Factory](#)

[Effects Core](#) (optional; for use with the `show` and `hide` options)

## Additional Notes:

This widget requires some functional CSS, otherwise it won't work. If you build a custom theme, use the widget's specific CSS file as a starting point.

## Options

### active

Type: [Boolean](#) or [Integer](#)

Default: `0`

Which panel is currently open.

#### Multiple types supported:

**Boolean:** Setting `active` to `false` will collapse all panels. This requires the `collapsible` option to be `true`.

**Integer:** The zero-based index of the panel that is active (open). A negative value selects panels going backward from the last panel.

#### Code examples:

Initialize the tabs with the active option specified:

```
1 | $( ".selector" ).tabs({ active: 1 });
```

Get or set the active option, after initialization:

```
1 | // getter
2 | var active = $( ".selector" ).tabs( "opt
3 |
4 | // setter
5 | $( ".selector" ).tabs( "option", "active
```

### collapsible

Type: [Boolean](#)

Default: `false`

When set to `true`, the active panel can be closed.

## Code examples:

Initialize the tabs with the collapsible option specified:

```
1 | $( ".selector" ).tabs({ collapsible: true });
```

Get or set the collapsible option, after initialization:

```
1 | // getter
2 | var collapsible = $( ".selector" ).tabs( "option"
3 |
4 | // setter
5 | $( ".selector" ).tabs( "option", "collapsible"
```

## disabled

Type: [Boolean](#) or [Array](#)

Default: `false`

Which tabs are disabled.

**Multiple types supported:**

**Boolean:** Enable or disable all tabs.

**Array:** An array containing the zero-based indexes of the tabs that should be disabled, e.g., `[ 0, 2 ]` would disable the first and third tab.

## Code examples:

Initialize the tabs with the disabled option specified:

```
1 | $( ".selector" ).tabs({ disabled: [ 0, 2 ] });
```

Get or set the disabled option, after initialization:

```
1 // getter
2 var disabled = $( ".selector" ).tabs( "option"
3
4 // setter
5 $( ".selector" ).tabs( "option", "disabled", [
```

## event

Type: [String](#)

Default: `"click"`

The type of event that the tabs should react to in order to activate the tab. To activate on hover, use `"mouseover"`.

### Code examples:

Initialize the tabs with the event option specified:

```
1 $( ".selector" ).tabs({ event: "mouseover" });
```

Get or set the event option, after initialization:

```
1 // getter
2 var event = $( ".selector" ).tabs( "option",
3
4 // setter
5 $( ".selector" ).tabs( "option", "event", "mouseover"
```

## heightStyle

Type: [String](#)

Default: `"content"`

Controls the height of the tabs widget and each panel.

Possible values:

"auto": All panels will be set to the height of the tallest panel.

"fill": Expand to the available height based on the tabs' parent height.

"content": Each panel will be only as tall as its content.

### Code examples:

Initialize the tabs with the heightStyle option specified:

```
1 | $( ".selector" ).tabs({ heightStyle: "fill" });
```

Get or set the heightStyle option, after initialization:

```
1 | // getter  
2 | var heightStyle = $( ".selector" ).tabs( "option"  
3 |  
4 | // setter  
5 | $( ".selector" ).tabs( "option", "heightStyle"
```

**hide** Type: Boolean or Number or String or Object

Default: `null`

If and how to animate the hiding of the panel.

**Multiple types supported:**

**Boolean:** When set to `false`, no animation will be used and the panel will be hidden immediately. When set to `true`, the panel will fade out with the default duration and the default easing.

**Number:** The panel will fade out with the specified duration and the default easing.

**String:** The panel will be hidden using the specified effect. The value can either be the name of a built-in

jQuery animation method, such as `"slideUp"`, or the name of a jQuery UI effect, such as `"fold"`. In either case the effect will be used with the default duration and the default easing.

**Object:** If the value is an object, then `effect`, `delay`, `duration`, and `easing` properties may be provided. If the `effect` property contains the name of a jQuery method, then that method will be used; otherwise it is assumed to be the name of a jQuery UI effect. When using a jQuery UI effect that supports additional settings, you may include those settings in the object and they will be passed to the effect. If `duration` or `easing` is omitted, then the default values will be used. If `effect` is omitted, then `"fadeOut"` will be used. If `delay` is omitted, then no delay is used.

### Code examples:

Initialize the tabs with the hide option specified:

```
1 | $( ".selector" ).tabs({ hide: { effect: "explode" }
```

Get or set the hide option, after initialization:

```
1 | // getter
2 | var hide = $( ".selector" ).tabs( "option", "hide"
3 |
4 | // setter
5 | $( ".selector" ).tabs( "option", "hide", { effect:
```

---

**show** Type: [Boolean](#) or [Number](#) or [String](#) or [Object](#)

Default: `null`

If and how to animate the showing of the panel.

**Multiple types supported:**

**Boolean:** When set to `false`, no animation will be used and the panel will be shown immediately. When set to `true`, the panel will fade in with the default duration and the default easing.

**Number:** The panel will fade in with the specified duration and the default easing.

**String:** The panel will be shown using the specified effect. The value can either be the name of a built-in jQuery animation method, such as `"slideDown"`, or the name of a jQuery UI effect, such as `"fold"`. In either case the effect will be used with the default duration and the default easing.

**Object:** If the value is an object, then `effect`, `delay`, `duration`, and `easing` properties may be provided. If the `effect` property contains the name of a jQuery method, then that method will be used; otherwise it is assumed to be the name of a jQuery UI effect. When using a jQuery UI effect that supports additional settings, you may include those settings in the object and they will be passed to the effect. If `duration` or `easing` is omitted, then the default values will be used. If `effect` is omitted, then `"fadeIn"` will be used. If `delay` is omitted, then no delay is used.

### Code examples:

Initialize the tabs with the show option specified:

```
1 | $( ".selector" ).tabs({ show: { effect: "blind" }
```

Get or set the show option, after initialization:

```
1 | // getter
2 | var show = $( ".selector" ).tabs( "option", "show"
3 |
4 | // setter
5 | $( ".selector" ).tabs( "option", "show", { effect:
```



## Methods

### destroy()

Removes the tabs functionality completely. This will return the element back to its pre-init state.

This method does not accept any arguments.

#### Code examples:

Invoke the destroy method:

```
1 | $( ".selector" ).tabs( "destroy" );
```

### disable()

Disables all tabs.

This method does not accept any arguments.

#### Code examples:

Invoke the method:

```
1 | $( ".selector" ).tabs( "disable" );
```

### disable( index )

Disables a tab. The selected tab cannot be disabled. To disable more than one tab at once, set the `disabled` option:

```
$( "#tabs" ).tabs( "option", "disabled", [ 1, 2, 3 ] ).
```

## index

Type: [Number](#) or [String](#)  
Which tab to disable.

### Code examples:

Invoke the method:

```
1 | $( ".selector" ).tabs( "disable", 1 );
```

## enable()

Enables all tabs.

This method does not accept any arguments.

### Code examples:

Invoke the method:

```
1 | $( ".selector" ).tabs( "enable" );
```

## enable( index )

Enables a tab. To enable more than one tab at once reset the disabled property like: `$( "#example" ).tabs( "option", "disabled", [ ] );`.

### index

Type: [Number](#) or [String](#)  
Which tab to enable.

### Code examples:

Invoke the method:

```
|
```

```
1 | $( ".selector" ).tabs( "enable", 1 );
```

## load( index )

Loads the panel content of a remote tab.

### index

Type: [Number](#) or [String](#)  
Which tab to load.

### Code examples:

Invoke the load method:

```
1 | $( ".selector" ).tabs( "load", 1 );
```

## option( optionName )

Returns: [Object](#)

Gets the value currently associated with the specified `optionName`.

### optionName

Type: [String](#)  
The name of the option to get.

### Code examples:

Invoke the method:

```
1 | var isDisabled = $( ".selector" ).tabs( "option"
```



## option()

Returns: [PlainObject](#)

Gets an object containing key/value pairs representing the current tabs options hash.

This method does not accept any arguments.

### Code examples:

Invoke the method:

```
1 | var options = $( ".selector" ).tabs( "option"
```

---

## option( optionName, value )

Sets the value of the tabs option associated with the specified `optionName`.

### optionName

Type: String

The name of the option to set.

### value

Type: Object

A value to set for the option.

### Code examples:

Invoke the method:

```
1 | $( ".selector" ).tabs( "option", "disabled",
```

---

## option( options )

Sets one or more options for the tabs.

### options

Type: Object

A map of option-value pairs to set.

### Code examples:

Invoke the method:

```
1 | $( ".selector" ).tabs( "option", { disabled:
```

---

## refresh()

Process any tabs that were added or removed directly in the DOM and recompute the height of the tab panels. Results depend on the content and the `heightStyle` option.

This method does not accept any arguments.

### Code examples:

Invoke the refresh method:

```
1 | $( ".selector" ).tabs( "refresh" );
```

---

## widget()

Returns: *jQuery*

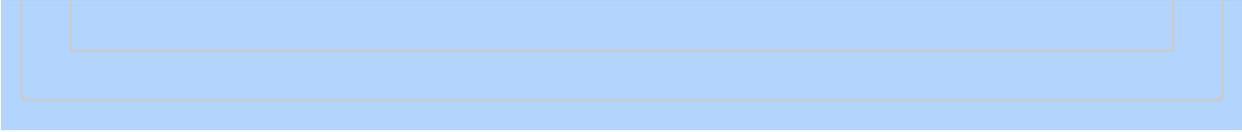
Returns a `jQuery` object containing the tabs container.

This method does not accept any arguments.

### Code examples:

Invoke the widget method:

```
1 | var widget = $( ".selector" ).tabs( "widget" );
```



## Events

**activate( event, ui )**

Type: `tabsactivate`

Triggered after a tab has been activated (after animation completes). If the tabs were previously collapsed, `ui.oldTab` and `ui.oldPanel` will be empty jQuery objects. If the tabs are collapsing, `ui.newTab` and `ui.newPanel` will be empty jQuery objects.

**event**

Type: [Event](#)

**ui**

Type: [Object](#)

**newTab**

Type: [jQuery](#)

The tab that was just activated.

**oldTab**

Type: [jQuery](#)

The tab that was just deactivated.

**newPanel**

Type: [jQuery](#)

The panel that was just activated.

**oldPanel**

Type: [jQuery](#)

The panel that was just deactivated.

### Code examples:

Initialize the tabs with the activate callback specified:

```
1 | $( ".selector" ).tabs({
```

```
2 | activate: function( event, ui ) {}  
3 | });
```

Bind an event listener to the tabsactivate event:

```
1 | $( ".selector" ).on( "tabsactivate", function
```

## beforeActivate( event, ui )

Type: `tabsbeforeactivate`

Triggered directly after a tab is activated. Can be canceled to prevent the tab from activating. If the tabs are currently collapsed, `ui.oldTab` and `ui.oldPanel` will be empty jQuery objects. If the tabs are collapsing, `ui.newTab` and `ui.newPanel` will be empty jQuery objects.

### event

Type: [Event](#)

### ui

Type: [Object](#)

#### newTab

Type: [jQuery](#)

The tab that is about to be activated.

#### oldTab

Type: [jQuery](#)

The tab that is about to be deactivated.

#### newPanel

Type: [jQuery](#)

The panel that is about to be activated.

#### oldPanel

Type: [jQuery](#)

The panel that is about to be deactivated.

### Code examples:

Initialize the tabs with the `beforeActivate` callback specified:

```
1 | $( ".selector" ).tabs({
2 |   beforeActivate: function( event, ui ) {}
3 | });
```

Bind an event listener to the `tabsbeforeactivate` event:

```
1 | $( ".selector" ).on( "tabsbeforeactivate", function
```

**beforeLoad( event, ui )**    *Type:* `tabsbeforeload`

Triggered when a remote tab is about to be loaded, after the `beforeActivate` event. Can be canceled to prevent the tab panel from loading content; though the panel will still be activated. This event is triggered just before the Ajax request is made, so modifications can be made to `ui.jqXHR` and `ui.ajaxSettings`.

*Note: Although `ui.ajaxSettings` is provided and can be modified, some of these settings have already been processed by jQuery. For example, `prefilters` have been applied, `data` has been processed, and `type` has been determined. The `beforeLoad` event occurs at the same time, and therefore has the same restrictions, as the `beforeSend` callback from `jQuery.ajax()`.*

**event**  
*Type:* `Event`

## ui

Type: `Object`

### tab

Type: `jQuery`

The tab that is being loaded.

### panel

Type: `jQuery`

The panel which will be populated by the Ajax response.

### jqXHR

Type: `jqXHR`

The `jqXHR` object that is requesting the content.

### ajaxSettings

Type: `Object`

The settings that will be used by `jQuery.ajax` to request the content.

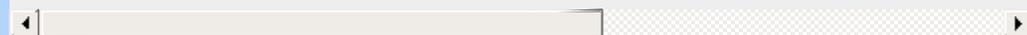
## Code examples:

Initialize the tabs with the `beforeLoad` callback specified:

```
1 | $( ".selector" ).tabs({  
2 |   beforeLoad: function( event, ui ) {}  
3 | });
```

Bind an event listener to the `tabsbeforeload` event:

```
1 | $( ".selector" ).on( "tabsbeforeload", function
```



---

`create( event, ui )`

Type: `tabscreate`

Triggered when the tabs are created. If the tabs are collapsed, `ui.tab` and `ui.panel` will be empty jQuery objects.

**event**

Type: [Event](#)

**ui**

Type: [Object](#)

**tab**

Type: [jQuery](#)

The active tab.

**panel**

Type: [jQuery](#)

The active panel.

### Code examples:

Initialize the tabs with the create callback specified:

```
1 | $( ".selector" ).tabs({  
2 |   create: function( event, ui ) {}  
3 | });
```

Bind an event listener to the tabscreate event:

```
1 | $( ".selector" ).on( "tabscreate", function( event, ui ) {
```

---

**load( event, ui )**

Type: `tabsload`

Triggered after a remote tab has been loaded.

**event**

Type: [Event](#)

## ui

Type: Object

### tab

Type: jQuery

The tab that was just loaded.

### panel

Type: jQuery

The panel which was just populated by the Ajax response.

## Code examples:

Initialize the tabs with the load callback specified:

```
1 | $( ".selector" ).tabs({  
2 |   load: function( event, ui ) {}  
3 | });
```

Bind an event listener to the tabsload event:

```
1 | $( ".selector" ).on( "tabsload", function( event, ui
```

## Example:

### *A simple jQuery UI Tabs*

```
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>tabs demo</title>
6   <link rel="stylesheet" href="http://code.jq
7   <script src="http://code.jquery.com/jquery-:
8   <script src="http://code.jquery.com/ui/1.10
9 </head>
10 <body>
11
12 <div id="tabs">
13   <ul>
14     <li><a href="#fragment-1"><span>One</span>
15     <li><a href="#fragment-2"><span>Two</span>
16     <li><a href="#fragment-3"><span>Three</sp
17   </ul>
18   <div id="fragment-1">
19     <p>First tab is active by default:</p>
20     <pre><code>$ ( "#tabs" ).tabs(); </code></
21   </div>
22   <div id="fragment-2">
23     Lorem ipsum dolor sit amet, consectetur
24     Lorem ipsum dolor sit amet, consectetur
25   </div>
26   <div id="fragment-3">
27     Lorem ipsum dolor sit amet, consectetur
28     Lorem ipsum dolor sit amet, consectetur
29     Lorem ipsum dolor sit amet, consectetur
30   </div>
31 </div>
```

```
32  
33 <script>  
34 $( "#tabs" ).tabs();  
35 </script>  
36  
37 </body>  
38 </html>
```

Demo

POWERED BYHERONOTE

A new version of this book is available!

---

# .toggle()

Categories: [Effects](#) | [Effects Core](#) | [Method Overrides](#) | [Methods](#)

```
.toggle( effect [, options ] [, duration ] [, complete ] )
```

Returns: `jQuery`

**Description:** *Display or hide the matched elements, using custom effects.*

```
.toggle( effect [, options ] [, duration ] [, complete ] )
```

**effect**

Type: `String`

A string indicating which `effect` to use for the transition.

**options**

Type: `Object`

Effect-specific settings and `easing`.

**duration** (default: `400`)

Type: `Number` or `String`

A string or number determining how long the animation will run.

**complete**

Type: `Function()`

A function to call once the animation is complete.

```
.toggle( options )
```

**options**

Type: `Object`

All animation settings. The only required property is `effect`.

**effect**

Type: `String`

A string indicating which `effect` to use for the transition.

**easing** (default: `"swing"`)

Type: `String`

A string indicating which `easing` function to use for the transition.

**duration** (default: `400`)

Type: `Number` or `String`

A string or number determining how long the animation will run.

---

**complete**

Type: `Function()`

A function to call once the animation is complete.

This plugin extends jQuery's built-in `.toggle()` method. If jQuery UI is not loaded, calling the `.toggle()` method may not fail directly, as the method still exists. However, the expected behavior will not occur.

## Example:

*Toggle a div using the fold effect.*

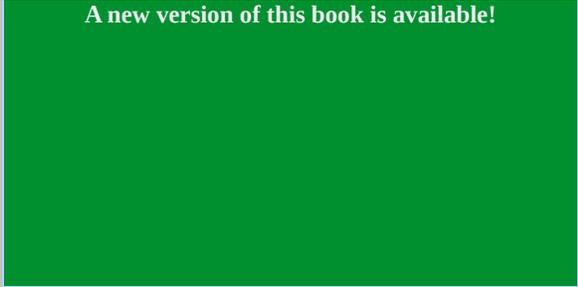
```
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>toggle demo</title>
6   <link rel="stylesheet" href="http://code.jq
7   <style>
8   div {
9     width: 100px;
10    height: 100px;
11    background: #ccc;
12    border: 1px solid #000;
13  }
14 </style>
15 <script src="http://code.jquery.com/jquery-:
16 <script src="http://code.jquery.com/ui/1.10
17 </head>
18 <body>
19
20 <button>toggle the div</button>
21 <div></div>
22
23 <script>
24 $( "button" ).click(function() {
25   $( "div" ).toggle( "fold", 1000 );
26 });
27 </script>
28
29 </body>
30 </html>
```



Demo

**POWERED BYHERONOTE**

A new version of this book is available!



---

# `.toggleClass()`

Categories: [Effects](#) | [Effects Core](#) | [Method Overrides](#)

```
.toggleClass( className [, switch ] [, duration ] [, easing ]  
[, complete ] ) Returns: jQuery
```

**Description:** Add or remove one or more classes from each element in the set of matched elements, depending on either the class's presence or the value of the switch argument, while animating all style changes.

```
.toggleClass( className [, switch ] [, duration ] [,  
easing ] [, complete ] )
```

**className**

Type: [String](#)

One or more class names (space separated) to be toggled for each element in the matched set.

**switch**

Type: [Boolean](#)

A boolean value to determine whether the class should be added or removed.

**duration** (default: `400`)

Type: [Number](#) or [String](#)

A string or number determining how long the animation will run.

**easing** (default: `swing`)

Type: [String](#)

A string indicating which [easing](#) function to use for the transition.

**complete**

Type: [Function\(\)](#)

A function to call once the animation is complete.

Similar to native CSS transitions, jQuery UI's class animations provide a smooth transition from one state to another while allowing you to keep all the details about which styles to change in CSS and out of your JavaScript. All class animation methods, including `.toggleClass()`, support custom durations and easings, as well as

providing a callback for when the animation completes.

Not all styles can be animated. For example, there is no way to animate a background image. Any styles that cannot be animated will be changed at the end of the animation.

This plugin extends jQuery's built-in `.toggleClass()` method. If jQuery UI is not loaded, calling the `.toggleClass()` method may not fail directly, as the method still exists. However, the expected behavior will not occur.

## Example:

*Toggle the class "big-blue" for the matched elements.*

```
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>toggleClass demo</title>
6   <link rel="stylesheet" href="http://code.jq
7   <style>
8   div {
9     width: 100px;
10    height: 100px;
11    background-color: #ccc;
12  }
13  .big-blue {
14    width: 200px;
15    height: 200px;
16    background-color: #00f;
17  }
18  </style>
19  <script src="http://code.jquery.com/jquery-:
20  <script src="http://code.jquery.com/ui/1.10
21 </head>
22 <body>
23
24 <div></div>
25
26 <script>
27 $( "div" ).click(function() {
28   $( this ).toggleClass( "big-blue", 1000, "e
29 });
30 </script>
```

```
31  
32 </body>  
33 </html>
```

Demo

POWERED BYHERONOTE

A new version of this book is available!

---

# Tooltip Widget

Categories: [Widgets](#)

---

Tooltip Widget

version added: 1.9

**Description:** *Customizable, themeable tooltips, replacing native tooltips.*

## QuickNav

## Examples

### Options

[content](#)  
[disabled](#)  
[hide](#)  
[items](#)  
[position](#)  
[show](#)  
[tooltipClass](#)  
[track](#)

### Methods

[close](#)  
[destroy](#)  
[disable](#)  
[enable](#)  
[open](#)  
[option](#)  
[widget](#)

### Events

[create](#)  
[open](#)  
[close](#)

Tooltip replaces native tooltips, making them themable as well as allowing various customizations:

- Display other content than just the title, like inline footnotes or extra content retrieved via Ajax.

- Customize the positioning, e.g., to center the tooltip above elements.

- Add extra styling to customize the appearance, for warning or error fields.

A fade animation is used by default to show and hide the tooltip, making the appearance a bit more organic, compared to just toggling the visibility. This can be customized with the [show](#) and [hide](#) options.

The [items](#) and [content](#) options need to stay in-sync. If you change one of them, you need to change the other.

In general, disabled elements do not trigger any DOM events. Therefore, it is not possible to properly control tooltips for disabled elements, since we need to listen to events to determine when to show and hide the tooltip. As a result, jQuery UI does not guarantee any level of support for tooltips attached to disabled elements. Unfortunately, this means that if you require tooltips on disabled

elements, you may end up with a mixture of native tooltips and jQuery UI tooltips.

## Dependencies

[UI Core](#)

[Widget Factory](#)

[Position](#)

[Effects Core](#) (optional; for use with the `show` and `hide` options)

## Additional Notes:

This widget requires some functional CSS, otherwise it won't work. If you build a custom theme, use the widget's specific CSS file as a starting point.



Default: `false`

Disables the tooltip if set to `true`.

### Code examples:

Initialize the tooltip with the disabled option specified:

```
1 | $( ".selector" ).tooltip({ disabled: true });
```

Get or set the disabled option, after initialization:

```
1 | // getter
2 | var disabled = $( ".selector" ).tooltip( "option"
3 |
4 | // setter
5 | $( ".selector" ).tooltip( "option", "disabled"
```

---

## hide Type: Boolean or Number or String or Object

Default: `null`

If and how to animate the hiding of the tooltip.

### Multiple types supported:

**Boolean:** When set to `false`, no animation will be used and the tooltip will be hidden immediately. When set to `true`, the tooltip will fade out with the default duration and the default easing.

**Number:** The tooltip will fade out with the specified duration and the default easing.

**String:** The tooltip will be hidden using the specified effect. The value can either be the name of a built-in jQuery animation method, such as `"slideUp"`, or the name of a jQuery UI effect, such as `"fold"`. In either case the effect will be used with the default duration and the default easing.

**Object:** If the value is an object, then `effect`, `delay`,

`duration`, and `easing` properties may be provided. If the `effect` property contains the name of a jQuery method, then that method will be used; otherwise it is assumed to be the name of a jQuery UI effect. When using a jQuery UI effect that supports additional settings, you may include those settings in the object and they will be passed to the effect. If `duration` or `easing` is omitted, then the default values will be used. If `effect` is omitted, then `"fadeOut"` will be used. If `delay` is omitted, then no delay is used.

### Code examples:

Initialize the tooltip with the hide option specified:

```
1 | $( ".selector" ).tooltip({ hide: { effect: "explode" }
```

Get or set the hide option, after initialization:

```
1 | // getter
2 | var hide = $( ".selector" ).tooltip( "option"
3 |
4 | // setter
5 | $( ".selector" ).tooltip( "option", "hide", { effect
```

## items

Type: [Selector](#)

Default: `[title]`

A selector indicating which items should show tooltips. Customize if you're using something other than the title attribute for the tooltip content, or if you need a different selector for event delegation.

*When changing this option, you likely need to also change the [content](#) option.*

### Code examples:

Initialize the tooltip with the items option specified:

```
1 | $( ".selector" ).tooltip({ items: "img[alt]" });
```

Get or set the items option, after initialization:

```
1 | // getter
2 | var items = $( ".selector" ).tooltip( "option"
3 |
4 | // setter
5 | $( ".selector" ).tooltip( "option", "items",
```

## position

Type: Object

**Default:** { my: "left top+15", at: "left bottom", collision: "flipfit" }

Identifies the position of the tooltip in relation to the associated target element. The `of` option defaults to the target element, but you can specify another element to position against. You can refer to the [jQuery UI Position](#) utility for more details about the various options.

### Code examples:

Initialize the tooltip with the position option specified:

```
1 | $( ".selector" ).tooltip({ position: { my: "left+15 c
```

Get or set the position option, after initialization:

```
1 | // getter
```

```
2 var position = $( ".selector" ).tooltip( "option"  
3  
4 // setter  
5 $( ".selector" ).tooltip( "option", "position"
```

**show** Type: [Boolean](#) or [Number](#) or [String](#) or [Object](#)

Default: `null`

If and how to animate the showing of the tooltip.

#### Multiple types supported:

**Boolean:** When set to `false`, no animation will be used and the tooltip will be shown immediately. When set to `true`, the tooltip will fade in with the default duration and the default easing.

**Number:** The tooltip will fade in with the specified duration and the default easing.

**String:** The tooltip will be shown using the specified effect. The value can either be the name of a built-in jQuery animation method, such as `"slideDown"`, or the name of a jQuery UI effect, such as `"fold"`. In either case the effect will be used with the default duration and the default easing.

**Object:** If the value is an object, then `effect`, `delay`, `duration`, and `easing` properties may be provided. If the `effect` property contains the name of a jQuery method, then that method will be used; otherwise it is assumed to be the name of a jQuery UI effect. When using a jQuery UI effect that supports additional settings, you may include those settings in the object and they will be passed to the effect. If `duration` or `easing` is omitted, then the default values will be used. If `effect` is omitted, then `"fadeIn"` will be used. If `delay` is omitted, then no delay is used.

#### Code examples:

Initialize the tooltip with the show option specified:

```
1 | $( ".selector" ).tooltip({ show: { effect: "blind"
```

Get or set the show option, after initialization:

```
1 | // getter
2 | var show = $( ".selector" ).tooltip( "option"
3 |
4 | // setter
5 | $( ".selector" ).tooltip( "option", "show", { effect
```

## tooltipClass

Type: [String](#)

Default: `null`

A class to add to the widget, can be used to display various tooltip types, like warnings or errors.

This may get replaced by the [classes option](#).

### Code examples:

Initialize the tooltip with the tooltipClass option specified:

```
1 | $( ".selector" ).tooltip({ tooltipClass: "custom-too
```

Get or set the tooltipClass option, after initialization:

```
1 | // getter
2 | var tooltipClass = $( ".selector" ).tooltip(
3 |
4 | // setter
5 | $( ".selector" ).tooltip( "option", "tooltipClass"
```

## track

Type: [Boolean](#)

Default: `false`

Whether the tooltip should track (follow) the mouse.

### Code examples:

Initialize the tooltip with the track option specified:

```
1 | $( ".selector" ).tooltip({ track: true });
```

Get or set the track option, after initialization:

```
1 | // getter
2 | var track = $( ".selector" ).tooltip( "option"
3 |
4 | // setter
5 | $( ".selector" ).tooltip( "option", "track",
```

## Methods

### close()

Closes a tooltip. This is only intended to be called for non-delegated tooltips.

This method does not accept any arguments.

#### Code examples:

Invoke the close method:

```
1 | $( ".selector" ).tooltip( "close" );
```

### destroy()

Removes the tooltip functionality completely. This will return the element back to its pre-init state.

This method does not accept any arguments.

#### Code examples:

Invoke the destroy method:

```
1 | $( ".selector" ).tooltip( "destroy" );
```

### disable()

Disables the tooltip.

This method does not accept any arguments.

### Code examples:

Invoke the disable method:

```
1 | $( ".selector" ).tooltip( "disable" );
```

---

## enable()

Enables the tooltip.

This method does not accept any arguments.

### Code examples:

Invoke the enable method:

```
1 | $( ".selector" ).tooltip( "enable" );
```

---

## open()

Programmatically open a tooltip. This is only intended to be called for non-delegated tooltips.

This method does not accept any arguments.

### Code examples:

Invoke the open method:

```
1 | $( ".selector" ).tooltip( "open" );
```

---

**option( optionName )**

Returns: *Object*

Gets the value currently associated with the specified `optionName`.

**optionName**

Type: String

The name of the option to get.

### Code examples:

Invoke the method:

```
1 | var isDisabled = $( ".selector" ).tooltip( "option"
```

## option()

Returns: PlainObject

Gets an object containing key/value pairs representing the current tooltip options hash.

This method does not accept any arguments.

### Code examples:

Invoke the method:

```
1 | var options = $( ".selector" ).tooltip( "option"
```

## option( optionName, value )

Sets the value of the tooltip option associated with the specified `optionName`.

**optionName**

Type: String

The name of the option to set.

## value

Type: [Object](#)

A value to set for the option.

### Code examples:

Invoke the method:

```
1 | $( ".selector" ).tooltip( "option", "disabled"
```

## option( options )

Sets one or more options for the tooltip.

### options

Type: [Object](#)

A map of option-value pairs to set.

### Code examples:

Invoke the method:

```
1 | $( ".selector" ).tooltip( "option", { disabled:
```

## widget()

Returns: [jQuery](#)

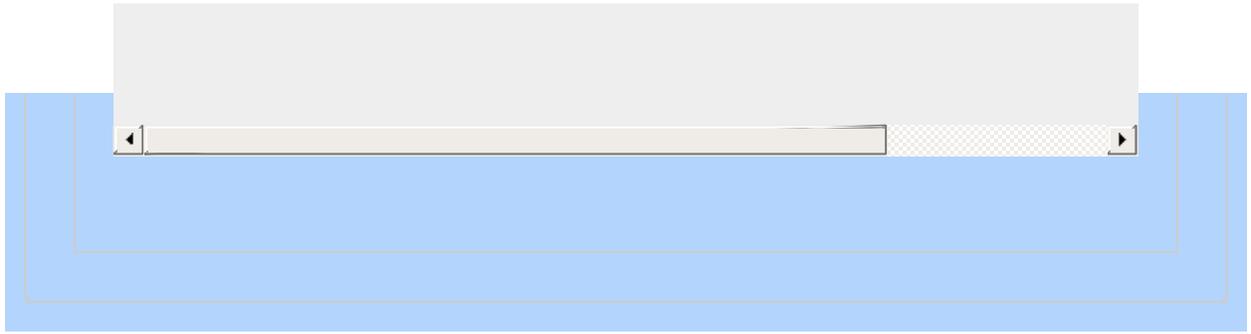
Returns a [jQuery](#) object containing the original element.

This method does not accept any arguments.

### Code examples:

Invoke the widget method:

```
1 | var widget = $( ".selector" ).tooltip( "widget"
```



## Events

### close( event, ui )

Type: tooltipclose

Triggered when a tooltip is closed, triggered on `focusout` or `mouseleave`.

#### event

Type: [Event](#)

#### ui

Type: [Object](#)

#### tooltip

Type: [jQuery](#)

The generated tooltip element.

### Code examples:

Initialize the tooltip with the close callback specified:

```
1 | $( ".selector" ).tooltip({  
2 |   close: function( event, ui ) {}  
3 | });
```

Bind an event listener to the tooltipclose event:

```
1 | $( ".selector" ).on( "tooltipclose", fui
```

### create( event, ui )

Type: tooltipcreate

Triggered when the tooltip is created.

**event**  
Type: [Event](#)

**ui**  
Type: [Object](#)

### Code examples:

Initialize the tooltip with the create callback specified:

```
1 | $( ".selector" ).tooltip({  
2 |   create: function( event, ui ) {}  
3 | });
```

Bind an event listener to the tooltipcreate event:

```
1 | $( ".selector" ).on( "tooltipcreate", function
```



**open( event, ui )** *Type:* `tooltipopen`

Triggered when a tooltip is shown, triggered on `focusin` or `mouseover`.

**event**  
Type: [Event](#)

**ui**  
Type: [Object](#)

**tooltip**  
Type: [jQuery](#)  
The generated tooltip element.

### Code examples:

Initialize the tooltip with the open callback specified:

```
1 | $( ".selector" ).tooltip({  
2 |   open: function( event, ui ) {}  
3 | });
```

Bind an event listener to the tooltipopen event:

```
1 | $( ".selector" ).on( "tooltipopen", function( event,
```



## Example:

*Create a tooltip on the document using event delegation for all elements with a title attribute.*

```
1  <!doctype html>
2  <html lang="en">
3  <head>
4      <meta charset="utf-8">
5      <title>tooltip demo</title>
6      <link rel="stylesheet" href="http://code.jq
7      <script src="http://code.jquery.com/jquery-:
8      <script src="http://code.jquery.com/ui/1.10
9  </head>
10 <body>
11
12 <p>
13     <a href="#" title="Anchor description">Anch
14     <input title="Input help">
15 </p>
16 <script>
17     $( document ).tooltip();
18 </script>
19
20 </body>
21 </html>
```

Demo

**A new version of this book is available!**

---

# Transfer Effect

Categories: [Effects](#)

## Transfer Effect

**Description:** *Transfers the outline of an element to another element*

### transfer

**className**

Type: String

argumental class name the transfer element will receive.

**to**

Type: String

jQuery selector, the element to transfer to.

Very useful when trying to visualize interaction between two elements.

The transfer element itself has the class `ui-effects-transfer`, and needs to be styled by you, for example by adding a background or border.

## Example:

*Clicking on the green element transfers to the other.*

```
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>transfer demo</title>
6   <link rel="stylesheet" href="http://code.jq
7   <style>
8   div.green {
9     width: 100px;
10    height: 80px;
11    background: green;
12    border: 1px solid black;
13    position: relative;
14  }
15  div.red {
16    margin-top: 10px;
17    width: 50px;
18    height: 30px;
19    background: red;
20    border: 1px solid black;
21    position: relative;
22  }
23  .ui-effects-transfer {
24    border: 1px dotted black;
25  }
26 </style>
27 <script src="http://code.jquery.com/jquery-:
28 <script src="http://code.jquery.com/ui/1.10
29 </head>
30 <body>
31
```

```
32 <div class="green"></div>
33 <div class="red"></div>
34
35 <script>
36 $( "div" ).click(function() {
37     var i = 1 - $( "div" ).index( this );
38     $( this ).effect( "transfer", { to: $( "div" ).eq( i
39 });
40 </script>
41
42 </body>
43 </html>
```

Demo

POWERED BYHERONOTE

A new version of this book is available!

---

# .uniqueId()

Categories: [Methods](#) | [UI Core](#)

`.uniqueId()`

Returns: `jQuery`

**Description:** *Generate and apply a unique id for the set of matched elements.*

`.uniqueId()`

version added: 1.9

This method does not accept any arguments.

Many widgets need to generate unique ids for elements. `.uniqueId()` will check if the element has an id, and if not, it will generate one and set it on the element. It is safe to call `.uniqueId()` on an element without checking if it already has an id. If/when the widget needs to clean up after itself, the `.removeUniqueId()` method will remove the id from the element if it was added by `.uniqueId()` and leave the id alone if it was not. `.removeUniqueId()` is able to be smart about this because the generated ids have a prefix of `"ui-id-"`.

POWERED BYHERONOTE

A new version of this book is available!

---

# .zIndex()

Categories: [Methods](#) | [UI Core](#)

## Contents:

[.zIndex\(\)](#)

[.zIndex\(\)](#)

[.zIndex\( zIndex \)](#)

[.zIndex\( zIndex \)](#)

`.zIndex()`

Returns: *jQuery*

**Description:** *Get the z-index for an element.*

**`.zIndex()`**

This method does not accept any arguments.

The `.zIndex()` method is useful for finding the z-index of an element, regardless of whether the z-index is set directly on the element or on one of its ancestors. In order to determine the z-index, this method will start at the specified element and walk up the DOM until it finds an element that is positioned and has a z-index set. If no such element is found, then the method will return a value of `0`.

This method assumes that elements with nested z-indexes would not have a z-index of `0`. For example, given the following DOM the inner element will be treated as if it does not have a z-index set because it is impossible to differentiate between an explicit value of `0` and no value in Internet Explorer.

```
1 | <div style="z-index: -10;">
2 |   <div style="z-index: 0;"></div>
3 | </div>
```

`.zIndex( zIndex )`

Returns: `Integer`

**Description:** *Set the z-index for an element.*

**.zIndex( zIndex )**

**zIndex**

Type: `Integer`

The z-index to set.

This is equivalent to `.css( "zIndex", zIndex )`.

**POWERED BYHERONOTE**

A new version of this book is available!

---

## Category: Effects

jQuery UI adds quite a bit of functionality on top of [jQuery's built-in effects](#). jQuery UI adds support for animating colors and class transitions, as well as providing several additional [easings](#). In addition, a full suite of custom effects are available for use when showing and hiding elements or just to add some visual appeal.

Also in: [Effects Core](#) | [Method Overrides](#)

---

## **.addClass()**

Adds the specified class(es) to each of the set of matched elements while animating all style changes.

---

## **Blind Effect**

The blind effect hides or shows an element by wrapping the element in a container, and “pulling the blinds”

---

## **Bounce Effect**

The bounce effect bounces an element. When used with hide or show, the last or first bounce will also fade in/out.

---

## Clip Effect

The clip effect will hide or show an element by clipping the element vertically or horizontally.

---

## Drop Effect

The drop effect hides or shows an element fading in/out and sliding in a direction.

Also in: [Effects Core](#) | [Methods](#)

---

## **.effect()**

Apply an animation effect to an element.

---

## **Explode Effect**

The explode effect hides or shows an element by splitting it into pieces.

---

## **Fade Effect**

The fade effect hides or shows an element by fading it.

---

## Fold Effect

The fold effect hides or shows an element by folding it.

Also in: [Effects Core](#) | [Method Overrides](#) | [Methods](#)

---

## **.hide()**

Hide the matched elements, using custom effects.

---

## Highlight Effect

The highlight effect hides or shows an element by animating its background color first.

---

## **Puff Effect**

Creates a puff effect by scaling the element up and hiding it at the same time.

---

## Pulsate Effect

The pulsate effect hides or shows an element by pulsing it in or out.

Also in: [Effects Core](#) | [Method Overrides](#)

---

## **.removeClass()**

Removes the specified class(es) from each of the set of matched elements while animating all style changes.

---

## **Scale Effect**

Shrink or grow an element by a percentage factor.

---

## Shake Effect

Shakes the element multiple times, vertically or horizontally.

Also in: [Effects Core](#) | [Method Overrides](#) | [Methods](#)

---

## **.show()**

Display the matched elements, using custom effects.

---

## Size Effect

Resize an element to a specified width and height.

---

## Slide Effect

Slides the element out of the viewport.

Also in: [Effects Core](#)

---

## **.switchClass()**

Adds and removes the specified class(es) to each of the set of matched elements while animating all style changes.

Also in: [Effects Core](#) | [Method Overrides](#) | [Methods](#)

---

## **.toggle()**

Display or hide the matched elements, using custom effects.

Also in: [Effects Core](#) | [Method Overrides](#)

---

## **.toggleClass()**

Add or remove one or more classes from each element in the set of matched elements, depending on either the class's presence or the value of the switch argument, while animating all style changes.

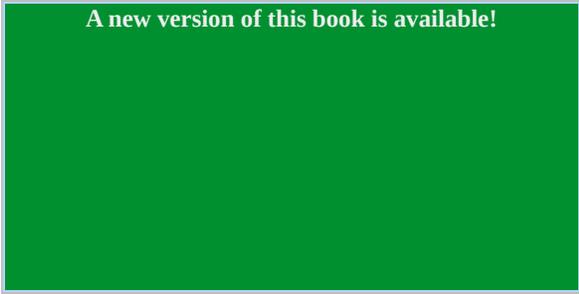
---

## Transfer Effect

Transfers the outline of an element to another element

POWERED BYHERONOTE

A new version of this book is available!



---

## Category: Effects Core

Functionality provided by `jquery.ui.effect.js`. In addition to the methods listed below, `jquery.ui.effect.js` also includes several [easings](#).

Also in: [Effects](#) | [Method Overrides](#)

---

## **.addClass()**

Adds the specified class(es) to each of the set of matched elements while animating all style changes.

Also in: [Effects](#) | [Methods](#)

---

## **.effect()**

Apply an animation effect to an element.

Also in: [Effects](#) | [Method Overrides](#) | [Methods](#)

---

## **.hide()**

Hide the matched elements, using custom effects.

Also in: [Effects](#) | [Method Overrides](#)

---

## **.removeClass()**

Removes the specified class(es) from each of the set of matched elements while animating all style changes.

Also in: [Effects](#) | [Method Overrides](#) | [Methods](#)

---

## **.show()**

Display the matched elements, using custom effects.

Also in: [Effects](#)

---

## **.switchClass()**

Adds and removes the specified class(es) to each of the set of matched elements while animating all style changes.

Also in: [Effects](#) | [Method Overrides](#) | [Methods](#)

---

## **.toggle()**

Display or hide the matched elements, using custom effects.

Also in: [Effects](#) | [Method Overrides](#)

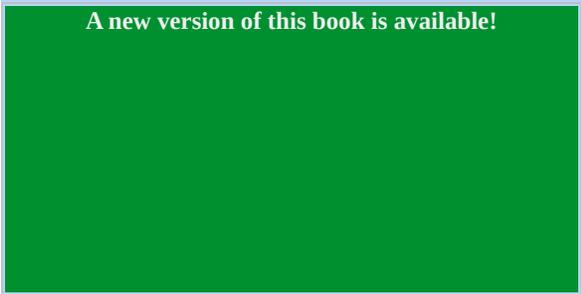
---

## **.toggleClass()**

Add or remove one or more classes from each element in the set of matched elements, depending on either the class's presence or the value of the switch argument, while animating all style changes.

**POWERED BYHERONOTE**

A new version of this book is available!



---

## Category: Interactions

jQuery UI provides a set of mouse-based interactions as building blocks for rich interfaces and complex widgets.

---

## **Draggable Widget**

Allow elements to be moved using the mouse.

---

## Draggable Widget

Create targets for draggable elements.

Also in: [Utilities](#)

---

## **Mouse Interaction**

The base interaction layer.

---

## **Resizable Widget**

Change the size of an element using the mouse.

---

## Selectable Widget

Use the mouse to select elements, individually or in a group.

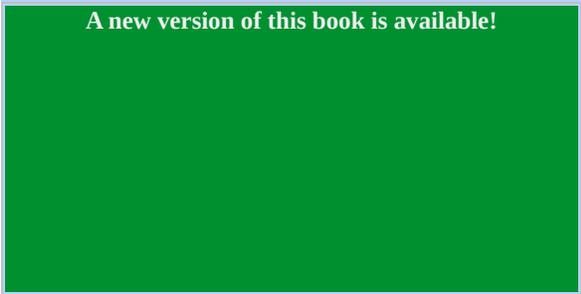
---

## Sortable Widget

Reorder elements in a list or grid using the mouse.

POWERED BYHERONOTE

A new version of this book is available!



---

## Category: Method Overrides

jQuery UI overrides several built-in jQuery methods in order to provide additional functionality. When using these overrides, it's important to make sure that jQuery UI is loaded. If jQuery UI is not loaded, the methods will still exist, but the expected functionality will not be available, resulting in bugs that may be hard to track down.

Also in: [Effects](#) | [Effects Core](#)

---

## **.addClass()**

Adds the specified class(es) to each of the set of matched elements while animating all style changes.

Also in: [Methods](#) | [UI Core](#)

---

## **.focus()**

Asynchronously set focus to an element.

Also in: [Effects](#) | [Effects Core](#) | [Methods](#)

---

## **.hide()**

Hide the matched elements, using custom effects.

Also in: [Methods](#) | [Utilities](#)

---

## **.position()**

Position an element relative to another.

Also in: [Effects](#) | [Effects Core](#)

---

## **.removeClass()**

Removes the specified class(es) from each of the set of matched elements while animating all style changes.

Also in: [Effects](#) | [Effects Core](#) | [Methods](#)

---

## **.show()**

Display the matched elements, using custom effects.

Also in: [Effects](#) | [Effects Core](#) | [Methods](#)

---

## **.toggle()**

Display or hide the matched elements, using custom effects.

Also in: [Effects](#) | [Effects Core](#)

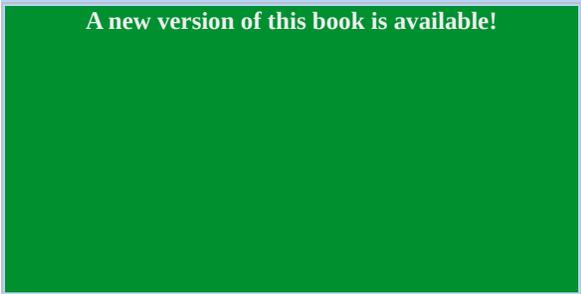
---

## **.toggleClass()**

Add or remove one or more classes from each element in the set of matched elements, depending on either the class's presence or the value of the switch argument, while animating all style changes.

**POWERED BYHERONOTE**

A new version of this book is available!



---

# Category: Methods

Although jQuery UI is mostly comprised of widgets, interactions, and effects, there are also a few simple methods that are added for convenience.

Also in: [UI Core](#)

---

## **.disableSelection()**

Disable selection of text content within the set of matched elements.

Also in: [Effects](#) | [Effects Core](#)

---

## **.effect()**

Apply an animation effect to an element.

Also in: [UI Core](#)

---

## **.enableSelection()**

Enable selection of text content within the set of matched elements.

Also in: [Method Overrides](#) | [UI Core](#)

---

## **.focus()**

Asynchronously set focus to an element.

Also in: [Effects](#) | [Effects Core](#) | [Method Overrides](#)

---

## **.hide()**

Hide the matched elements, using custom effects.

Also in: [Method Overrides](#) | [Utilities](#)

---

## **.position()**

Position an element relative to another.

Also in: [UI Core](#)

---

## **.removeUniqueId()**

Remove ids that were set by `.uniqueId()` for the set of matched elements.

Also in: [UI Core](#)

---

## **.scrollParent()**

Get the closest ancestor element that is scrollable.

Also in: [Effects](#) | [Effects Core](#) | [Method Overrides](#)

---

## **.show()**

Display the matched elements, using custom effects.

Also in: [Effects](#) | [Effects Core](#) | [Method Overrides](#)

---

## **.toggle()**

Display or hide the matched elements, using custom effects.

Also in: [UI Core](#)

---

## **.uniqueId()**

Generate and apply a unique id for the set of matched elements.

Also in: [UI Core](#)

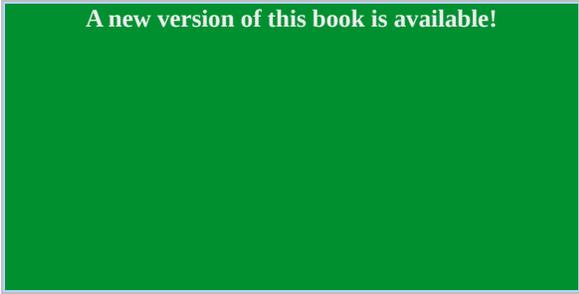
---

## **.zIndex()**

Get the z-index for an element.

**POWERED BYHERONOTE**

A new version of this book is available!



---

# Category: Selectors

Also in: [UI Core](#)

---

## **:data() Selector**

Selects elements which have data stored under the specified key.

Also in: [UI Core](#)

---

## **:focusable Selector**

Selects elements which can be focused.

Also in: [UI Core](#)

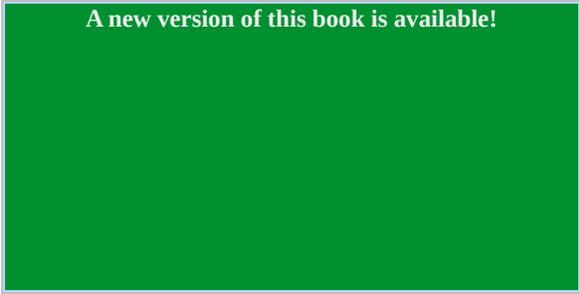
---

## **:tabbable Selector**

Selects elements which the user can focus via tabbing.

**POWERED BYHERONOTE**

**A new version of this book is available!**



---

# Category: UI Core

Functionality provided by jquery.ui.core.js.

Also in: [Selectors](#)

---

## **:data() Selector**

Selects elements which have data stored under the specified key.

Also in: [Methods](#)

---

## **.disableSelection()**

Disable selection of text content within the set of matched elements.

Also in: [Methods](#)

---

## **.enableSelection()**

Enable selection of text content within the set of matched elements.

Also in: [Method Overrides](#) | [Methods](#)

---

## **.focus()**

Asynchronously set focus to an element.

Also in: [Selectors](#)

---

## **:focusable Selector**

Selects elements which can be focused.

Also in: [Methods](#)

---

## **.removeUniqueId()**

Remove ids that were set by `.uniqueId()` for the set of matched elements.

Also in: [Methods](#)

---

## **.scrollParent()**

Get the closest ancestor element that is scrollable.

Also in: [Selectors](#)

---

## **:tabbable Selector**

Selects elements which the user can focus via tabbing.

Also in: [Methods](#)

---

## **.uniqueId()**

Generate and apply a unique id for the set of matched elements.

Also in: [Methods](#)

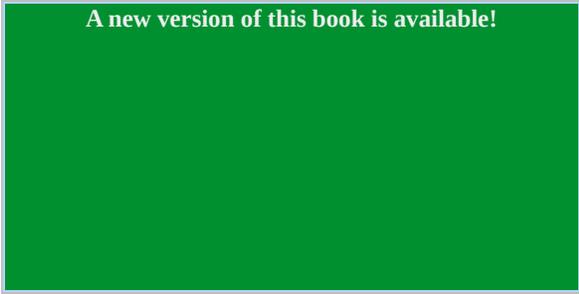
---

## **.zIndex()**

Get the z-index for an element.

**POWERED BYHERONOTE**

A new version of this book is available!



---

# Category: Utilities

Also in: [Widgets](#)

---

## Widget Factory

Create stateful jQuery plugins using the same abstraction as all jQuery UI widgets.

Also in: [Interactions](#)

---

## Mouse Interaction

The base interaction layer.

Also in: [Method Overrides](#) | [Methods](#)

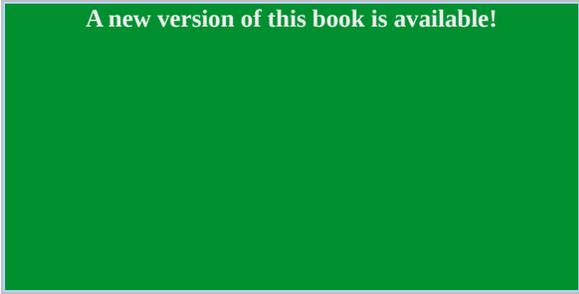
---

## **.position()**

Position an element relative to another.

**POWERED BYHERONOTE**

A new version of this book is available!



---

## Category: Widgets

Widgets are feature-rich, stateful plugins that have a full life-cycle, along with methods and events. Check out the [widget factory](#) documentation for more details.

---

## **Accordion Widget**

Convert a pair of headers and content panels into an accordion.

---

## **Autocomplete Widget**

Autocomplete enables users to quickly find and select from a pre-populated list of values as they type, leveraging searching and filtering.

---

## Button Widget

Themable buttons and button sets.

---

## Datepicker Widget

Select a date from a popup or inline calendar

---

## Dialog Widget

Open content in an interactive overlay.

Also in: [Utilities](#)

---

## Widget Factory

Create stateful jQuery plugins using the same abstraction as all jQuery UI widgets.

---

## **Menu Widget**

Themeable menu with mouse and keyboard interactions for navigation.

---

## **Progressbar Widget**

Display status of a determinate or indeterminate process.

---

## Slider Widget

Drag a handle to select a numeric value.

---

## Spinner Widget

Enhance a text input for entering numeric values, with up/down buttons and arrow key handling.

---

## **Tabs Widget**

A single content area with multiple panels, each associated with a header in a list.

---

## Tooltip Widget

Customizable, themeable tooltips, replacing native tooltips.

POWERED BYHERONOTE

A new version of this book is available!

---

# Easings

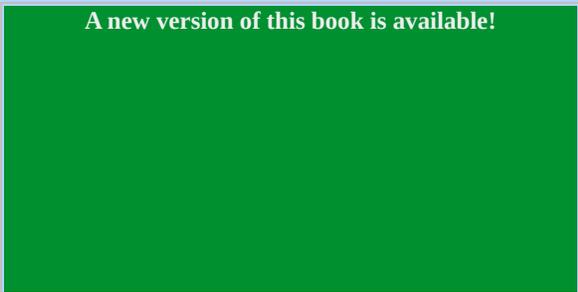
Easing functions specify the speed at which an animation progresses at different points within the animation. jQuery core ships with two easings: `linear`, which progresses at a constant pace throughout the animation, and `swing` (jQuery core's default easing), which progresses slightly slower at the beginning and end of the animation than it does in the middle of the animation. jQuery UI provides several additional easing functions, ranging from variations on the swing behavior to customized effects such as bouncing.

Some easings will result in negative values during the animation. Depending on the properties that are being animated, the actual value may be clamped at zero. For example, you can animate `left` to a negative value, but you cannot animate `height` or `opacity` to a negative value.

The best way to understand how an easing will affect an animation is to see the equation graphed over time. See below for a graph of all animations available in jQuery UI.

POWERED BYHERONOTE

A new version of this book is available!



# jQuery & jQuery UI Documentation

---

Book Name: jQuery

Version: 1.9.1

Updated: February 26,2013

Check for Updates: <http://www.HelpMesh.net/s/jquery/>

e-Book Update Manager:

[Download](#)