

# jQuery 1.1 Documentation

## Core

### **`$( expr, context )`**

---

This function accepts a string containing a CSS or basic XPath selector which is then used to match a set of elements. The core functionality of jQuery centers around this function. Everything in jQuery is based upon this, or uses this in some way. The most basic use of this function is to pass in an expression (usually consisting of CSS or XPath), which then finds all matching elements. By default, `$( )` looks for DOM elements within the context of the current HTML document.

#### **Parameters**

**expr:** *String*: An expression to search with

**context:** *Element|jQuery*: (optional) A DOM Element, Document or jQuery to use as context

#### **Returns**

jQuery

#### **Example**

Finds all p elements that are children of a div element.

```
$("#div > p")
```

#### **Before**

```
<p>one</p> <div><p>two</p></div> <p>three</p>
```

#### **Result**

```
[ <p>two</p> ]
```

#### **Example**

Searches for all inputs of type radio within the first form in the document

```
$("#input:radio", document.forms[0])
```

#### **Example**

This finds all div elements within the specified XML document.

```
$("#div", xml.responseXML)
```

#### **See Also**

`$(Element)`  
`$(Element<Array>)`

### **`$( html )`**

---

Create DOM elements on-the-fly from the provided String of raw HTML.

#### **Parameters**

**html:** *String*: A string of HTML to create on the fly.

#### **Returns**

jQuery

#### **Example**

Creates a div element (and all of its contents) dynamically, and appends it to the element with the ID of body. Internally, an element is created and its innerHTML property set to the given markup. It is therefore both quite flexible and limited.

```
$("#<div><p>Hello</p></div>").appendTo("#body")
```

### **`$( elems )`**

---

Wrap jQuery functionality around a single or multiple DOM Element(s). This function also accepts XML Documents and

Window objects as valid arguments (even though they are not DOM Elements).

**Parameters**

**elems:** *Element|Array<Element>*: DOM element(s) to be encapsulated by a jQuery object.

**Returns**

jQuery

**Example**

Same as \$("div > p") because the document

```
$(document).find("div > p")
```

**Before**

```
<p>one</p> <div><p>two</p></div> <p>three</p>
```

**Result**

```
[ <p>two</p> ]
```

**Example**

Sets the background color of the page to black.

```
$(document.body).background( "black" );
```

**Example**

Hides all the input elements within a form

```
$( myForm.elements ).hide()
```

---

## \$( fn )

A shorthand for \$(document).ready(), allowing you to bind a function to be executed when the DOM document has finished loading. This function behaves just like \$(document).ready(), in that it should be used to wrap all of the other \$() operations on your page. While this function is, technically, chainable - there really isn't much use for chaining against it. You can have as many \$(document).ready events on your page as you like. See ready(Function) for details about the ready event.

**Parameters**

**fn:** *Function*: The function to execute when the DOM is ready.

**Returns**

jQuery

**Example**

Executes the function when the DOM is ready to be used.

```
$(function(){ // Document is ready });
```

---

## \$( obj )

A means of creating a cloned copy of a jQuery object. This function copies the set of matched elements from one jQuery object and creates another, new, jQuery object containing the same elements.

**Parameters**

**obj:** *jQuery*: The jQuery object to be cloned.

**Returns**

jQuery

**Example**

Locates all p elements with all div elements, without disrupting the original jQuery object contained in 'div' (as would normally be the case if a simple div.find("p") was done).

```
var div = $("div"); $( div ).find("p");
```

---

## jquery( )

The current version of jQuery.

**Returns**

String

## length( )

---

The number of elements currently matched.

**Returns**

Number

**Example**

```
$("#img").length;
```

**Before**

```
 
```

**Result**

2

## size( )

---

The number of elements currently matched.

**Returns**

Number

**Example**

```
$("#img").size();
```

**Before**

```
 
```

**Result**

2

## get( )

---

Access all matched elements. This serves as a backwards-compatible way of accessing all matched elements (other than the jQuery object itself, which is, in fact, an array of elements).

**Returns**

Array<Element>

**Example**

Selects all images in the document and returns the DOM Elements as an Array

```
$("#img").get();
```

**Before**

```
 
```

**Result**

```
[   ]
```

## get( num )

---

Access a single matched element. num is used to access the Nth element matched.

**Parameters**

**num**: *Number*: Access the element in the Nth position.

**Returns**

Element

### **Example**

Selects all images in the document and returns the first one

```
$("#img").get(0);
```

### **Before**

```
 
```

### **Result**

```
[  ]
```

## **set( elems )**

---

Set the jQuery object to an array of elements, while maintaining the stack.

### **Parameters**

**elems**: *Elements*: An array of elements

### **Returns**

jQuery

### **Example**

```
$("#img").set([ document.body ]);
```

### **Result**

```
$("#img").set() == [ document.body ]
```

## **setArray( elems )**

---

Set the jQuery object to an array of elements. This operation is completely destructive - be sure to use `.set()` if you wish to maintain the jQuery stack.

### **Parameters**

**elems**: *Elements*: An array of elements

### **Returns**

jQuery

### **Example**

```
$("#img").setArray([ document.body ]);
```

### **Result**

```
$("#img").setArray() == [ document.body ]
```

## **each( fn )**

---

Execute a function within the context of every matched element. This means that every time the passed-in function is executed (which is once for every element matched) the 'this' keyword points to the specific element. Additionally, the function, when executed, is passed a single argument representing the position of the element in the matched set.

### **Parameters**

**fn**: *Function*: A function to execute

### **Returns**

jQuery

### **Example**

Iterates over two images and sets their src property

```
$("#img").each(function(i){ this.src = "test" + i + ".jpg"; });
```

### **Before**

```
<img/><img/>
```

### **Result**



## index( subject )

---

Searches every matched element for the object and returns the index of the element, if found, starting with zero. Returns -1 if the object wasn't found.

### Parameters

**subject:** *Element*: Object to search for

### Returns

Number

### Example

Returns the index for the element with ID foobar

```
$("#*").index( $('#foobar')[0] )
```

### Before

```
<div id="foobar"></div><b></b><span id="foo"></span>
```

### Result

0

### Example

Returns the index for the element with ID foo

```
$("#*").index( $('#foo'))
```

### Before

```
<div id="foobar"></div><b></b><span id="foo"></span>
```

### Result

2

### Example

Returns -1, as there is no element with ID bar

```
$("#*").index( $('#bar'))
```

### Before

```
<div id="foobar"></div><b></b><span id="foo"></span>
```

### Result

-1

## domManip( args, table, dir, fn )

---

### Parameters

**args:** *Array*:

**table:** *Boolean*: Insert TBODY in TABLEs if one is not found.

**dir:** *Number*: If dir<0, process args in reverse order.

**fn:** *Function*: The function doing the DOM manipulation.

### Returns

jQuery

## \$.extend( prop )

---

Extends the jQuery object itself. Can be used to add functions into the jQuery namespace and to add plugin methods (plugins).

### Parameters

**prop**: *Object*: The object that will be merged into the jQuery object

### Returns

Object

### Example

Adds two plugin methods.

```
jQuery.fn.extend({ check: function() { return this.each(function() { this.checked = true; }); }, uncheck: function() { return this.each(function() { this.checked = false; }); } }); $("input[@type=checkbox]").check();  
$("input[@type=radio]").uncheck();
```

### Example

Adds two functions into the jQuery namespace

```
jQuery.extend({ min: function(a, b) { return a < b ? a : b; }, max: function(a, b) { return a > b ? a : b; } });
```

## \$.noConflict( )

---

Run this function to give control of the \$ variable back to whichever library first implemented it. This helps to make sure that jQuery doesn't conflict with the \$ object of other libraries. By using this function, you will only be able to access jQuery using the 'jQuery' variable. For example, where you used to do \$("div p"), you now must do jQuery("div p").

### Returns

undefined

### Example

Maps the original object that was referenced by \$ back to \$

```
jQuery.noConflict(); // Do something with jQuery jQuery("div p").hide(); // Do something with another library's $()  
$("content").style.display = 'none';
```

### Example

Reverts the \$ alias and then creates and executes a function to provide the \$ as a jQuery alias inside the functions scope. Inside the function the original \$ object is not available. This works well for most plugins that don't rely on any other library.

```
jQuery.noConflict(); (function($){ $(function(){ // more code using $ as alias to jQuery }); })(jQuery); // other code  
using $ as an alias to the other library
```

## eq( pos )

---

Reduce the set of matched elements to a single element. The position of the element in the set of matched elements starts at 0 and goes to length - 1.

### Parameters

**pos**: *Number*: The index of the element that you wish to limit to.

### Returns

jQuery

### Example

```
$("p").eq(1)
```

#### Before

```
<p>This is just a test.</p><p>So is this</p>
```

#### Result

```
[ <p>So is this</p> ]
```

## lt( pos )

---

Reduce the set of matched elements to all elements before a given position. The position of the element in the set of matched elements starts at 0 and goes to length - 1.

### Parameters

**pos**: *Number*: Reduce the set to all elements below this position.

### Returns

jQuery

**Example**

```
$("#p").lt(1)
```

**Before**

```
<p>This is just a test.</p><p>So is this</p>
```

**Result**

```
[ <p>This is just a test.</p> ]
```

---

## gt( pos )

Reduce the set of matched elements to all elements after a given position. The position of the element in the set of matched elements starts at 0 and goes to length - 1.

**Parameters**

**pos:** *Number*: Reduce the set to all elements after this position.

**Returns**

jQuery

**Example**

```
$("#p").gt(0)
```

**Before**

```
<p>This is just a test.</p><p>So is this</p>
```

**Result**

```
[ <p>So is this</p> ]
```

---

## \$.find( )

**Returns**

Array<Element>

# DOM

---

## Attributes

---

### attr( name )

Access a property on the first matched element. This method makes it easy to retrieve a property value from the first matched element.

**Parameters**

**name:** *String*: The name of the property to access.

**Returns**

Object

**Example**

Returns the src attribute from the first image in the document.

```
$("#img").attr("src");
```

**Before**

```

```

**Result**

test.jpg

## attr( properties )

---

Set a key/value object as properties to all matched elements. This serves as the best way to set a large number of properties on all matched elements.

### Parameters

**properties:** *Map*: Key/value pairs to set as object properties.

### Returns

jQuery

### Example

Sets src and alt attributes to all images.

```
$("#img").attr({ src: "test.jpg", alt: "Test Image" });
```

### Before

```
<img/>
```

### Result

```

```

## attr( key, value )

---

Set a single property to a value, on all matched elements. Can compute values provided as `${formula}`, see second example. Note that you can't set the name property of input elements in IE. Use `$(html)` or `.append(html)` or `.html(html)` to create elements on the fly including the name property.

### Parameters

**key:** *String*: The name of the property to set.

**value:** *Object*: The value to set the property to.

### Returns

jQuery

### Example

Sets src attribute to all images.

```
$("#img").attr("src","test.jpg");
```

### Before

```
<img/>
```

### Result

```

```

### Example

Sets title attribute from src attribute, a shortcut for `attr(String,Function)`

```
$("#img").attr("title", "${this.src}");
```

### Before

```

```

### Result

```

```

## attr( key, value )

---

Set a single property to a computed value, on all matched elements. Instead of a value, a function is provided, that computes the value.

### Parameters

**key:** *String*: The name of the property to set.

**value:** *Function:* A function returning the value to set.

**Returns**

jQuery

**Example**

Sets title attribute from src attribute.

```
$("#img").attr("title", function() { return this.src });
```

**Before**

```

```

**Result**

```

```

## text( )

---

Get the text contents of all matched elements. The result is a string that contains the combined text contents of all matched elements. This method works on both HTML and XML documents.

**Returns**

String

**Example**

Gets the concatenated text of all paragraphs

```
$("#p").text();
```

**Before**

```
<p><b>Test</b> Paragraph.</p><p>Paraparagraph</p>
```

**Result**

```
Test Paragraph.Paraparagraph
```

## text( val )

---

Set the text contents of all matched elements. This has the same effect as `html()`.

**Parameters**

**val:** *String:* The text value to set the contents of the element to.

**Returns**

String

**Example**

Sets the text of all paragraphs.

```
$("#p").text("Some new text.");
```

**Before**

```
<p>Test Paragraph.</p>
```

**Result**

```
<p>Some new text.</p>
```

## val( )

---

Get the current value of the first matched element.

**Returns**

String

**Example**

```
$("#input").val();
```

**Before**

```
<input type="text" value="some text"/>
```

**Result**

```
"some text"
```

## val( val )

---

Set the value of every matched element.

**Parameters**

**val:** *String*: Set the property to the specified value.

**Returns**

jQuery

**Example**

```
$("#input").val("test");
```

**Before**

```
<input type="text" value="some text"/>
```

**Result**

```
<input type="text" value="test"/>
```

## html( )

---

Get the html contents of the first matched element. This property is not available on XML documents.

**Returns**

String

**Example**

```
$("#div").html();
```

**Before**

```
<div><input/></div>
```

**Result**

```
<input/>
```

## html( val )

---

Set the html contents of every matched element. This property is not available on XML documents.

**Parameters**

**val:** *String*: Set the html contents to the specified value.

**Returns**

jQuery

**Example**

```
$("#div").html("<b>new stuff</b>");
```

**Before**

```
<div><input/></div>
```

**Result**

```
<div><b>new stuff</b></div>
```

## removeAttr( name )

---

Remove an attribute from each of the matched elements.

**Parameters**

**name:** *String*: The name of the attribute to remove.

**Returns**

jQuery

**Example**

```
$("#input").removeAttr("disabled")
```

**Before**

```
<input disabled="disabled"/>
```

**Result**

```
<input/>
```

## addClass( class )

---

Adds the specified class to each of the set of matched elements.

**Parameters**

**class:** *String*: A CSS class to add to the elements

**Returns**

jQuery

**Example**

```
$("#p").addClass("selected")
```

**Before**

```
<p>Hello</p>
```

**Result**

```
[ <p class="selected">Hello</p> ]
```

## removeClass( class )

---

Removes all or the specified class from the set of matched elements.

**Parameters**

**class:** *String*: (optional) A CSS class to remove from the elements

**Returns**

jQuery

**Example**

```
$("#p").removeClass()
```

**Before**

```
<p class="selected">Hello</p>
```

**Result**

```
[ <p>Hello</p> ]
```

**Example**

```
$("#p").removeClass("selected")
```

**Before**

```
<p class="selected first">Hello</p>
```

**Result**

```
[ <p class="first">Hello</p> ]
```

## toggleClass( class )

---

Adds the specified class if it is not present, removes it if it is present.

**Parameters**

**class:** *String*: A CSS class with which to toggle the elements

**Returns**

jQuery

**Example**

```
$("#p").toggleClass("selected")
```

**Before**

```
<p>Hello</p><p class="selected">Hello Again</p>
```

**Result**

```
[ <p class="selected">Hello</p>, <p>Hello Again</p> ]
```

## Manipulation

---

### wrap( html )

---

Wrap all matched elements with a structure of other elements. This wrapping process is most useful for injecting additional structure into a document, without ruining the original semantic qualities of a document. This works by going through the first element provided (which is generated, on the fly, from the provided HTML) and finds the deepest ancestor element within its structure - it is that element that will en-wrap everything else. This does not work with elements that contain text. Any necessary text must be added after the wrapping is done.

**Parameters**

**html:** *String*: A string of HTML, that will be created on the fly and wrapped around the target.

**Returns**

jQuery

**Example**

```
$("#p").wrap("<div class='wrap'></div>");
```

**Before**

```
<p>Test Paragraph.</p>
```

**Result**

```
<div class='wrap'><p>Test Paragraph.</p></div>
```

### wrap( elem )

---

Wrap all matched elements with a structure of other elements. This wrapping process is most useful for injecting additional structure into a document, without ruining the original semantic qualities of a document. This works by going through the first element provided and finding the deepest ancestor element within its structure - it is that element that will en-wrap everything else. This does not work with elements that contain text. Any necessary text must be added after the wrapping is done.

**Parameters**

**elem:** *Element*: A DOM element that will be wrapped around the target.

**Returns**

jQuery

**Example**

```
$("#p").wrap( document.getElementById('content') );
```

**Before**

```
<p>Test Paragraph.</p><div id="content"></div>
```

**Result**

```
<div id="content"><p>Test Paragraph.</p></div>
```

### append( content )

---

Append content to the inside of every matched element. This operation is similar to doing an `appendChild` to all the specified elements, adding them into the document.

#### **Parameters**

**content:** `<Content>`: Content to append to the target

#### **Returns**

jQuery

#### **Example**

Appends some HTML to all paragraphs.

```
$("#p").append("<b>Hello</b>");
```

#### **Before**

```
<p>I would like to say: </p>
```

#### **Result**

```
<p>I would like to say: <b>Hello</b></p>
```

#### **Example**

Appends an Element to all paragraphs.

```
$("#p").append( $("#foo")[0] );
```

#### **Before**

```
<p>I would like to say: </p><b id="foo">Hello</b>
```

#### **Result**

```
<p>I would like to say: <b id="foo">Hello</b></p>
```

#### **Example**

Appends a jQuery object (similar to an Array of DOM Elements) to all paragraphs.

```
$("#p").append( $("#b" ) );
```

#### **Before**

```
<p>I would like to say: </p><b>Hello</b>
```

#### **Result**

```
<p>I would like to say: <b>Hello</b></p>
```

#### **See Also**

```
prepend(<Content>)  
before(<Content>)  
after(<Content>)
```

## prepend( content )

---

Prepend content to the inside of every matched element. This operation is the best way to insert elements inside, at the beginning, of all matched elements.

#### **Parameters**

**content:** `<Content>`: Content to prepend to the target.

#### **Returns**

jQuery

#### **Example**

Prepends some HTML to all paragraphs.

```
$("#p").prepend("<b>Hello</b>");
```

#### **Before**

```
<p>I would like to say: </p>
```

#### **Result**

```
<p><b>Hello</b>I would like to say: </p>
```

#### **Example**

Prepends an Element to all paragraphs.

```
$("#p").prepend( $("#foo")[0] );
```

**Before**

<p>I would like to say: </p><b id="foo">Hello</b>

**Result**

<p><b id="foo">Hello</b>I would like to say: </p>

**Example**

Prepends a jQuery object (similar to an Array of DOM Elements) to all paragraphs.

```
$("p").prepend( $("b" ) );
```

**Before**

<p>I would like to say: </p><b>Hello</b>

**Result**

<p><b>Hello</b>I would like to say: </p>

**See Also**

append(<Content>)  
before(<Content>)  
after(<Content>)

## before( content )

---

Insert content before each of the matched elements.

**Parameters**

**content:** <Content>: Content to insert before each target.

**Returns**

jQuery

**Example**

Inserts some HTML before all paragraphs.

```
$("p").before("<b>Hello</b>");
```

**Before**

<p>I would like to say: </p>

**Result**

<b>Hello</b><p>I would like to say: </p>

**Example**

Inserts an Element before all paragraphs.

```
$("p").before( $("#foo")[0] );
```

**Before**

<p>I would like to say: </p><b id="foo">Hello</b>

**Result**

<b id="foo">Hello</b><p>I would like to say: </p>

**Example**

Inserts a jQuery object (similar to an Array of DOM Elements) before all paragraphs.

```
$("p").before( $("b" ) );
```

**Before**

<p>I would like to say: </p><b>Hello</b>

**Result**

<b>Hello</b><p>I would like to say: </p>

**See Also**

append(<Content>)  
prepend(<Content>)  
after(<Content>)

## after( content )

---

Insert content after each of the matched elements.

### Parameters

**content**: *<Content>*: Content to insert after each target.

### Returns

jQuery

### Example

Inserts some HTML after all paragraphs.

```
$("p").after("<b>Hello</b>");
```

#### Before

```
<p>I would like to say: </p>
```

#### Result

```
<p>I would like to say: </p><b>Hello</b>
```

### Example

Inserts an Element after all paragraphs.

```
$("p").after( $("#foo")[0] );
```

#### Before

```
<b id="foo">Hello</b><p>I would like to say: </p>
```

#### Result

```
<p>I would like to say: </p><b id="foo">Hello</b>
```

### Example

Inserts a jQuery object (similar to an Array of DOM Elements) after all paragraphs.

```
$("p").after( $("b" ) );
```

#### Before

```
<b>Hello</b><p>I would like to say: </p>
```

#### Result

```
<p>I would like to say: </p><b>Hello</b>
```

### See Also

```
append(<Content>)  
prepend(<Content>)  
before(<Content>)
```

## clone( )

---

Clone matched DOM Elements and select the clones. This is useful for moving copies of the elements to another location in the DOM.

### Returns

jQuery

### Example

Clones all b elements (and selects the clones) and prepends them to all paragraphs.

```
$("b").clone().prependTo("p");
```

#### Before

```
<b>Hello</b><p>, how are you?</p>
```

#### Result

```
<b>Hello</b><p><b>Hello</b>, how are you?</p>
```

## appendTo( expr )

---

Append all of the matched elements to another, specified, set of elements. This operation is, essentially, the reverse of doing a regular `$(A).append(B)`, in that instead of appending B to A, you're appending A to B.

**Parameters**

**expr:** *String:* A jQuery expression of elements to match.

**Returns**

jQuery

**Example**

Appends all paragraphs to the element with the ID "foo"

```
$("p").appendTo("#foo");
```

**Before**

```
<p>I would like to say: </p><div id="foo"></div>
```

**Result**

```
<div id="foo"><p>I would like to say: </p></div>
```

## prependTo( expr )

---

Prepend all of the matched elements to another, specified, set of elements. This operation is, essentially, the reverse of doing a regular `$(A).prepend(B)`, in that instead of prepending B to A, you're prepending A to B.

**Parameters**

**expr:** *String:* A jQuery expression of elements to match.

**Returns**

jQuery

**Example**

Prepends all paragraphs to the element with the ID "foo"

```
$("p").prependTo("#foo");
```

**Before**

```
<p>I would like to say: </p><div id="foo"><b>Hello</b></div>
```

**Result**

```
<div id="foo"><p>I would like to say: </p><b>Hello</b></div>
```

## insertBefore( expr )

---

Insert all of the matched elements before another, specified, set of elements. This operation is, essentially, the reverse of doing a regular `$(A).before(B)`, in that instead of inserting B before A, you're inserting A before B.

**Parameters**

**expr:** *String:* A jQuery expression of elements to match.

**Returns**

jQuery

**Example**

Same as `$("#foo").before("p")`

```
$("p").insertBefore("#foo");
```

**Before**

```
<div id="foo">Hello</div><p>I would like to say: </p>
```

**Result**

```
<p>I would like to say: </p><div id="foo">Hello</div>
```

## insertAfter( expr )

---

Insert all of the matched elements after another, specified, set of elements. This operation is, essentially, the reverse of

doing a regular `$(A).after(B)`, in that instead of inserting B after A, you're inserting A after B.

**Parameters**

**expr:** *String*: A jQuery expression of elements to match.

**Returns**

jQuery

**Example**

Same as `$("#foo").after("p")`

```
$("#p").insertAfter("#foo");
```

**Before**

```
<p>I would like to say: </p><div id="foo">Hello</div>
```

**Result**

```
<div id="foo">Hello</div><p>I would like to say: </p>
```

---

## remove( expr )

Removes all matched elements from the DOM. This does NOT remove them from the jQuery object, allowing you to use the matched elements further. Can be filtered with an optional expressions.

**Parameters**

**expr:** *String*: (optional) A jQuery expression to filter elements by.

**Returns**

jQuery

**Example**

```
$("#p").remove();
```

**Before**

```
<p>Hello</p> how are <p>you?</p>
```

**Result**

```
how are
```

**Example**

```
$("#p").remove(".hello");
```

**Before**

```
<p class="hello">Hello</p> how are <p>you?</p>
```

**Result**

```
how are <p>you?</p>
```

---

## empty( )

Removes all child nodes from the set of matched elements.

**Returns**

jQuery

**Example**

```
$("#p").empty();
```

**Before**

```
<p>Hello, <span>Person</span> <a href="#">and person</a></p>
```

**Result**

```
[ <p></p> ]
```

---

## Traversing

## end( )

---

End the most recent 'destructive' operation, reverting the list of matched elements back to its previous state. After an end operation, the list of matched elements will revert to the last state of matched elements. If there was no destructive operation before, an empty set is returned.

### Returns

jQuery

### Example

Selects all paragraphs, finds span elements inside these, and reverts the selection back to the paragraphs.

```
$("#p").find("span").end();
```

### Before

```
<p><span>Hello</span>, how are you?</p>
```

### Result

```
[ <p>...</p> ]
```

## find( expr )

---

Searches for all elements that match the specified expression. This method is a good way to find additional descendant elements with which to process. All searching is done using a jQuery expression. The expression can be written using CSS 1-3 Selector syntax, or basic XPath.

### Parameters

**expr:** *String*: An expression to search with.

### Returns

jQuery

### Example

Starts with all paragraphs and searches for descendant span elements, same as \$("#p span")

```
$("#p").find("span");
```

### Before

```
<p><span>Hello</span>, how are you?</p>
```

### Result

```
[ <span>Hello</span> ]
```

## filter( expression )

---

Removes all elements from the set of matched elements that do not match the specified expression(s). This method is used to narrow down the results of a search. Provide a String array of expressions to apply multiple filters at once.

### Parameters

**expression:** *String|Array<String>*: Expression(s) to search with.

### Returns

jQuery

### Example

Selects all paragraphs and removes those without a class "selected".

```
$("#p").filter(".selected")
```

### Before

```
<p class="selected">Hello</p><p>How are you?</p>
```

### Result

```
[ <p class="selected">Hello</p> ]
```

### Example

Selects all paragraphs and removes those without class "selected" and being the first one.

```
$("#p").filter([".selected", ":first"])
```

**Before**

```
<p>Hello</p><p>Hello Again</p><p class="selected">And Again</p>
```

**Result**

```
[ <p>Hello</p>, <p class="selected">And Again</p> ]
```

## filter( filter )

---

Removes all elements from the set of matched elements that do not pass the specified filter. This method is used to narrow down the results of a search.

**Parameters**

**filter**: *Function*: A function to use for filtering

**Returns**

jQuery

**Example**

Remove all elements that have a child of element

```
$("#p").filter(function(index) { return $("#ol", this).length == 0; })
```

**Before**

```
<p><ol><li>Hello</li></ol></p><p>How are you?</p>
```

**Result**

```
[ <p>How are you?</p> ]
```

## not( el )

---

Removes the specified Element from the set of matched elements. This method is used to remove a single Element from a jQuery object.

**Parameters**

**el**: *Element*: An element to remove from the set

**Returns**

jQuery

**Example**

Removes the element with the ID "selected" from the set of all paragraphs.

```
$("#p").not( $("#selected")[0] )
```

**Before**

```
<p>Hello</p><p id="selected">Hello Again</p>
```

**Result**

```
[ <p>Hello</p> ]
```

## not( expr )

---

Removes elements matching the specified expression from the set of matched elements. This method is used to remove one or more elements from a jQuery object.

**Parameters**

**expr**: *String*: An expression with which to remove matching elements

**Returns**

jQuery

**Example**

Removes the element with the ID "selected" from the set of all paragraphs.

```
$("#p").not("#selected")
```

**Before**

```
<p>Hello</p><p id="selected">Hello Again</p>
```

**Result**

```
[ <p>Hello</p> ]
```

## add( expr )

---

Adds the elements matched by the expression to the jQuery object. This can be used to concatenate the result sets of two expressions.

**Parameters**

**expr:** *String*: An expression whose matched elements are added

**Returns**

jQuery

**Example**

```
$("p").add("span")
```

**Before**

```
<p>Hello</p><p><span>Hello Again</span></p>
```

**Result**

```
[ <p>Hello</p>, <span>Hello Again</span> ]
```

## add( elements )

---

Adds one or more Elements to the set of matched elements. This is used to add a set of Elements to a jQuery object.

**Parameters**

**elements:** *Element|Array<Element>*: One or more Elements to add

**Returns**

jQuery

**Example**

```
$("p").add( document.getElementById("a") )
```

**Before**

```
<p>Hello</p><p><span id="a">Hello Again</span></p>
```

**Result**

```
[ <p>Hello</p>, <span id="a">Hello Again</span> ]
```

**Example**

```
$("p").add([document.getElementById("a"), document.getElementById("b")])
```

**Before**

```
<p>Hello</p><p><span id="a">Hello Again</span><span id="b">And Again</span></p>
```

**Result**

```
[ <p>Hello</p>, <span id="a">Hello Again</span>, <span id="b">And Again</span> ]
```

## is( expr )

---

Checks the current selection against an expression and returns true, if at least one element of the selection fits the given expression. Does return false, if no element fits or the expression is not valid. filter(String) is used internally, therefore all rules that apply there apply here, too.

**Parameters**

**expr:** *String*: The expression with which to filter

**Returns**

Boolean

**Example**

Returns true, because the parent of the input is a form element

```
$("#input[@type='checkbox']").parent().is("form")
```

**Before**

```
<form><input type="checkbox" /></form>
```

**Result**

true

**Example**

Returns false, because the parent of the input is a p element

```
$("#input[@type='checkbox']").parent().is("form")
```

**Before**

```
<form><p><input type="checkbox" /></p></form>
```

**Result**

false

## parent( expr )

---

Get a set of elements containing the unique parents of the matched set of elements. Can be filtered with an optional expressions.

**Parameters**

**expr:** *String*: (optional) An expression to filter the parents with

**Returns**

jQuery

**Example**

Find the parent element of each paragraph.

```
$("#p").parent()
```

**Before**

```
<div><p>Hello</p><p>Hello</p></div>
```

**Result**

```
[ <div><p>Hello</p><p>Hello</p></div> ]
```

**Example**

Find the parent element of each paragraph with a class "selected".

```
$("#p").parent(".selected")
```

**Before**

```
<div><p>Hello</p></div><div class="selected"><p>Hello Again</p></div>
```

**Result**

```
[ <div class="selected"><p>Hello Again</p></div> ]
```

## parents( expr )

---

Get a set of elements containing the unique ancestors of the matched set of elements (except for the root element). Can be filtered with an optional expressions.

**Parameters**

**expr:** *String*: (optional) An expression to filter the ancestors with

**Returns**

jQuery

**Example**

Find all parent elements of each span.

```
$("#span").parents()
```

**Before**

```
<html><body><div><p><span>Hello</span></p><span>Hello Again</span></div></body></html>
```

**Result**

```
[ <body>...</body>, <div>...</div>, <p><span>Hello</span></p> ]
```

**Example**

Find all parent elements of each span that is a paragraph.

```
$("#span").parents("p")
```

**Before**

```
<html><body><div><p><span>Hello</span></p><span>Hello Again</span></div></body></html>
```

**Result**

```
[ <p><span>Hello</span></p> ]
```

## next( expr )

---

Get a set of elements containing the unique next siblings of each of the matched set of elements. It only returns the very next sibling, not all next siblings. Can be filtered with an optional expressions.

**Parameters**

**expr:** *String*: (optional) An expression to filter the next Elements with

**Returns**

jQuery

**Example**

Find the very next sibling of each paragraph.

```
$("#p").next()
```

**Before**

```
<p>Hello</p><p>Hello Again</p><div><span>And Again</span></div>
```

**Result**

```
[ <p>Hello Again</p>, <div><span>And Again</span></div> ]
```

**Example**

Find the very next sibling of each paragraph that has a class "selected".

```
$("#p").next(".selected")
```

**Before**

```
<p>Hello</p><p class="selected">Hello Again</p><div><span>And Again</span></div>
```

**Result**

```
[ <p class="selected">Hello Again</p> ]
```

## prev( expr )

---

Get a set of elements containing the unique previous siblings of each of the matched set of elements. Can be filtered with an optional expressions. It only returns the immediately previous sibling, not all previous siblings.

**Parameters**

**expr:** *String*: (optional) An expression to filter the previous Elements with

**Returns**

jQuery

**Example**

Find the very previous sibling of each paragraph.

```
$("#p").prev()
```

**Before**

```
<p>Hello</p><div><span>Hello Again</span></div><p>And Again</p>
```

**Result**

```
[ <div><span>Hello Again</span></div> ]
```

### Example

Find the very previous sibling of each paragraph that has a class "selected".

```
$("#p").prev(".selected")
```

### Before

```
<div><span>Hello</span></div><p class="selected">Hello Again</p><p>And Again</p>
```

### Result

```
[ <div><span>Hello</span></div> ]
```

## siblings( expr )

---

Get a set of elements containing all of the unique siblings of each of the matched set of elements. Can be filtered with an optional expressions.

### Parameters

**expr:** *String*: (optional) An expression to filter the sibling Elements with

### Returns

jQuery

### Example

Find all siblings of each div.

```
$("#div").siblings()
```

### Before

```
<p>Hello</p><div><span>Hello Again</span></div><p>And Again</p>
```

### Result

```
[ <p>Hello</p>, <p>And Again</p> ]
```

### Example

Find all siblings with a class "selected" of each div.

```
$("#div").siblings(".selected")
```

### Before

```
<div><span>Hello</span></div><p class="selected">Hello Again</p><p>And Again</p>
```

### Result

```
[ <p class="selected">Hello Again</p> ]
```

## children( expr )

---

Get a set of elements containing all of the unique children of each of the matched set of elements. Can be filtered with an optional expressions.

### Parameters

**expr:** *String*: (optional) An expression to filter the child Elements with

### Returns

jQuery

### Example

Find all children of each div.

```
$("#div").children()
```

### Before

```
<p>Hello</p><div><span>Hello Again</span></div><p>And Again</p>
```

### Result

```
[ <span>Hello Again</span> ]
```

### Example

Find all children with a class "selected" of each div.

```
$("#div").children(".selected")
```

**Before**

```
<div><span>Hello</span><p class="selected">Hello Again</p><p>And Again</p></div>
```

**Result**

```
[ <p class="selected">Hello Again</p> ]
```

## contains( str )

---

Filter the set of elements to those that contain the specified text.

**Parameters**

**str:** *String*: The string that will be contained within the text of an element.

**Returns**

jQuery

**Example**

```
$("#p").contains("test")
```

**Before**

```
<p>This is just a test.</p><p>So is this</p>
```

**Result**

```
[ <p>This is just a test.</p> ]
```

## \$.parents( elem )

---

All ancestors of a given element.

**Parameters**

**elem:** *Element*: The element to find the ancestors of.

**Returns**

Array<Element>

## \$.nth( cur, num, dir )

---

A handy, and fast, way to traverse in a particular direction and find a specific element.

**Parameters**

**cur:** *DOMElement*: The element to search from.

**num:** *Number|String*: The Nth result to match. Can be a number or a string (like 'even' or 'odd').

**dir:** *String*: The direction to move in (pass in something like 'previousSibling' or 'nextSibling').

**Returns**

DOMElement

## \$.sibling( elem )

---

All elements on a specified axis.

**Parameters**

**elem:** *Element*: The element to find all the siblings of (including itself).

**Returns**

Array

# CSS

## css( name )

---

Access a style property on the first matched element. This method makes it easy to retrieve a style property value from the first matched element.

### Parameters

**name:** *String*: The name of the property to access.

### Returns

String

### Example

Retrieves the color style of the first paragraph

```
$("#p").css("color");
```

### Before

```
<p style="color:red;">Test Paragraph.</p>
```

### Result

```
"red"
```

### Example

Retrieves the font-weight style of the first paragraph.

```
$("#p").css("font-weight");
```

### Before

```
<p style="font-weight: bold;">Test Paragraph.</p>
```

### Result

```
"bold"
```

## css( properties )

---

Set a key/value object as style properties to all matched elements. This serves as the best way to set a large number of style properties on all matched elements.

### Parameters

**properties:** *Map*: Key/value pairs to set as style properties.

### Returns

jQuery

### Example

Sets color and background styles to all p elements.

```
$("#p").css({ color: "red", background: "blue" });
```

### Before

```
<p>Test Paragraph.</p>
```

### Result

```
<p style="color:red; background:blue;">Test Paragraph.</p>
```

## css( key, value )

---

Set a single style property to a value, on all matched elements.

### Parameters

**key:** *String*: The name of the property to set.

**value:** *Object*: The value to set the property to.

**Returns**

jQuery

**Example**

Changes the color of all paragraphs to red

```
$("#p").css("color","red");
```

**Before**

```
<p>Test Paragraph.</p>
```

**Result**

```
<p style="color:red;">Test Paragraph.</p>
```

# JavaScript

## **\$.extend( target, prop1, propN )**

---

Extend one object with one or more others, returning the original, modified, object. This is a great utility for simple inheritance.

**Parameters**

**target:** *Object*: The object to extend

**prop1:** *Object*: The object that will be merged into the first.

**propN:** *Object*: (optional) More objects to merge into the first

**Returns**

Object

**Example**

Merge settings and options, modifying settings

```
var settings = { validate: false, limit: 5, name: "foo" }; var options = { validate: true, name: "bar" };  
jQuery.extend(settings, options);
```

**Result**

```
settings == { validate: true, limit: 5, name: "bar" }
```

**Example**

Merge defaults and options, without modifying the defaults

```
var defaults = { validate: false, limit: 5, name: "foo" }; var options = { validate: true, name: "bar" }; var settings =  
jQuery.extend({}, defaults, options);
```

**Result**

```
settings == { validate: true, limit: 5, name: "bar" }
```

## **\$.each( obj, fn )**

---

A generic iterator function, which can be used to seamlessly iterate over both objects and arrays. This function is not the same as `$.each()` - which is used to iterate, exclusively, over a jQuery object. This function can be used to iterate over anything. The callback has two arguments: the key (objects) or index (arrays) as first the first, and the value as the second.

**Parameters**

**obj:** *Object*: The object, or array, to iterate over.

**fn:** *Function*: The function that will be executed on every object.

**Returns**

Object

**Example**

This is an example of iterating over the items in an array, accessing both the current item and its index.

```
$.each( [0,1,2], function(i, n){ alert( "Item #" + i + ": " + n ); });
```

#### **Example**

This is an example of iterating over the properties in an Object, accessing both the current item and its key.

```
$.each( { name: "John", lang: "JS" }, function(i, n){ alert( "Name: " + i + ", Value: " + n ); });
```

## **\$.trim( str )**

---

Remove the whitespace from the beginning and end of a string.

#### **Parameters**

**str:** *String*: The string to trim.

#### **Returns**

String

#### **Example**

```
$.trim(" hello, how are you? ");
```

#### **Result**

```
"hello, how are you?"
```

## **\$.merge( first, second )**

---

Merge two arrays together, removing all duplicates. The new array is: All the results from the first array, followed by the unique results from the second array.

#### **Parameters**

**first:** *Array*: The first array to merge.

**second:** *Array*: The second array to merge.

#### **Returns**

Array

#### **Example**

Merges two arrays, removing the duplicate 2

```
$.merge( [0,1,2], [2,3,4] )
```

#### **Result**

```
[0,1,2,3,4]
```

#### **Example**

Merges two arrays, removing the duplicates 3 and 2

```
$.merge( [3,2,1], [4,3,2] )
```

#### **Result**

```
[3,2,1,4]
```

## **\$.grep( array, fn, inv )**

---

Filter items out of an array, by using a filter function. The specified function will be passed two arguments: The current array item and the index of the item in the array. The function must return 'true' to keep the item in the array, false to remove it.

#### **Parameters**

**array:** *Array*: The Array to find items in.

**fn:** *Function*: The function to process each item against.

**inv:** *Boolean*: Invert the selection - select the opposite of the function.

#### **Returns**

Array

**Example**

```
$.grep( [0,1,2], function(i){ return i > 0; });
```

**Result**

```
[1, 2]
```

## \$.map( array, fn )

---

Translate all items in an array to another array of items. The translation function that is provided to this method is called for each item in the array and is passed one argument: The item to be translated. The function can then return the translated value, 'null' (to remove the item), or an array of values - which will be flattened into the full array.

**Parameters**

**array:** *Array:* The Array to translate.

**fn:** *Function:* The function to process each item against.

**Returns**

Array

**Example**

Maps the original array to a new one and adds 4 to each value.

```
$.map( [0,1,2], function(i){ return i + 4; });
```

**Result**

```
[4, 5, 6]
```

**Example**

Maps the original array to a new one and adds 1 to each value if it is bigger then zero, otherwise it's removed-

```
$.map( [0,1,2], function(i){ return i > 0 ? i + 1 : null; });
```

**Result**

```
[2, 3]
```

**Example**

Maps the original array to a new one, each element is added with it's original value and the value plus one.

```
$.map( [0,1,2], function(i){ return [ i, i + 1 ]; });
```

**Result**

```
[0, 1, 1, 2, 2, 3]
```

## \$.browser( )

---

Contains flags for the useragent, read from navigator.userAgent. Available flags are: safari, opera, msie, mozilla This property is available before the DOM is ready, therefore you can use it to add ready events only for certain browsers. There are situations where object detections is not reliable enough, in that cases it makes sense to use browser detection. Simply try to avoid both! A combination of browser and object detection yields quite reliable results.

**Returns**

Boolean

**Example**

Returns true if the current useragent is some version of microsoft's internet explorer

```
$.browser.msie
```

**Example**

Alerts "this is safari!" only for safari browsers

```
if($.browser.safari) { $( function() { alert("this is safari!"); } ); }
```

## Effects

---

## show( )

---

Displays each of the set of matched elements if they are hidden.

**Returns**

jQuery

**Example**

```
$("#p").show()
```

**Before**

```
<p style="display: none">Hello</p>
```

**Result**

```
[ <p style="display: block">Hello</p> ]
```

---

## hide( )

---

Hides each of the set of matched elements if they are shown.

**Returns**

jQuery

**Example**

```
$("#p").hide()
```

**Before**

```
<p>Hello</p>
```

**Result**

```
[ <p style="display: none">Hello</p> ] var pass = true, div = $("#div"); div.hide().each(function(){ if ( this.style.display != "none" ) pass = false; }); ok( pass, "Hide" );
```

---

## toggle( )

---

Toggles each of the set of matched elements. If they are shown, toggle makes them hidden. If they are hidden, toggle makes them shown.

**Returns**

jQuery

**Example**

```
$("#p").toggle()
```

**Before**

```
<p>Hello</p><p style="display: none">Hello Again</p>
```

**Result**

```
[ <p style="display: none">Hello</p>, <p style="display: block">Hello Again</p> ]
```

---

## show( )

---

Displays each of the set of matched elements if they are hidden.

**Returns**

jQuery

**Example**

```
$("#p").show()
```

**Before**

```
<p style="display: none">Hello</p>
```

## Result

[ <p style="display: block">Hello</p> ]

## show( speed, callback )

---

Show all matched elements using a graceful animation and firing an optional callback after completion. The height, width, and opacity of each of the matched elements are changed dynamically according to the specified speed.

### Parameters

**speed**: *String|Number*: A string representing one of the three predefined speeds ("slow", "normal", or "fast") or the number of milliseconds to run the animation (e.g. 1000).

**callback**: *Function*: (optional) A function to be executed whenever the animation completes.

### Returns

jQuery

### Example

```
$("#p").show("slow");
```

### Example

```
$("#p").show("slow",function(){ alert("Animation Done."); });
```

## hide( )

---

Hides each of the set of matched elements if they are shown.

### Returns

jQuery

### Example

```
$("#p").hide()
```

### Before

```
<p>Hello</p>
```

### Result

[ <p style="display: none">Hello</p> ]

## hide( speed, callback )

---

Hide all matched elements using a graceful animation and firing an optional callback after completion. The height, width, and opacity of each of the matched elements are changed dynamically according to the specified speed.

### Parameters

**speed**: *String|Number*: A string representing one of the three predefined speeds ("slow", "normal", or "fast") or the number of milliseconds to run the animation (e.g. 1000).

**callback**: *Function*: (optional) A function to be executed whenever the animation completes.

### Returns

jQuery

### Example

```
$("#p").hide("slow");
```

### Example

```
$("#p").hide("slow",function(){ alert("Animation Done."); });
```

## toggle( )

---

Toggles each of the set of matched elements. If they are shown, toggle makes them hidden. If they are hidden, toggle makes them shown.

**Returns**

jQuery

**Example**

```
$("#p").toggle()
```

**Before**

```
<p>Hello</p><p style="display: none">Hello Again</p>
```

**Result**

```
[ <p style="display: none">Hello</p>, <p style="display: block">Hello Again</p> ]
```

## slideDown( speed, callback )

---

Reveal all matched elements by adjusting their height and firing an optional callback after completion. Only the height is adjusted for this animation, causing all matched elements to be revealed in a "sliding" manner.

**Parameters**

**speed**: *String|Number*: (optional) A string representing one of the three predefined speeds ("slow", "normal", or "fast") or the number of milliseconds to run the animation (e.g. 1000).

**callback**: *Function*: (optional) A function to be executed whenever the animation completes.

**Returns**

jQuery

**Example**

```
$("#p").slideDown("slow");
```

**Example**

```
$("#p").slideDown("slow",function(){ alert("Animation Done."); });
```

**See Also**

slideUp(String|Number,Function)  
slideToggle(String|Number,Function)

## slideUp( speed, callback )

---

Hide all matched elements by adjusting their height and firing an optional callback after completion. Only the height is adjusted for this animation, causing all matched elements to be hidden in a "sliding" manner.

**Parameters**

**speed**: *String|Number*: (optional) A string representing one of the three predefined speeds ("slow", "normal", or "fast") or the number of milliseconds to run the animation (e.g. 1000).

**callback**: *Function*: (optional) A function to be executed whenever the animation completes.

**Returns**

jQuery

**Example**

```
$("#p").slideUp("slow");
```

**Example**

```
$("#p").slideUp("slow",function(){ alert("Animation Done."); });
```

**See Also**

slideDown(String|Number,Function)  
slideToggle(String|Number,Function)

## slideToggle( speed, callback )

---

Toggle the visibility of all matched elements by adjusting their height and firing an optional callback after completion. Only the height is adjusted for this animation, causing all matched elements to be hidden in a "sliding" manner.

**Parameters**

**speed**: *String|Number*: (optional) A string representing one of the three predefined speeds ("slow", "normal", or "fast") or the number of milliseconds to run the animation (e.g. 1000).

**callback**: *Function*: (optional) A function to be executed whenever the animation completes.

**Returns**

jQuery

**Example**

```
$("#p").slideToggle("slow");
```

**Example**

```
$("#p").slideToggle("slow",function(){ alert("Animation Done."); });
```

**See Also**

slideDown(String|Number,Function)  
slideUp(String|Number,Function)

## fadeIn( speed, callback )

---

Fade in all matched elements by adjusting their opacity and firing an optional callback after completion. Only the opacity is adjusted for this animation, meaning that all of the matched elements should already have some form of height and width associated with them.

**Parameters**

**speed**: *String|Number*: (optional) A string representing one of the three predefined speeds ("slow", "normal", or "fast") or the number of milliseconds to run the animation (e.g. 1000).

**callback**: *Function*: (optional) A function to be executed whenever the animation completes.

**Returns**

jQuery

**Example**

```
$("#p").fadeIn("slow");
```

**Example**

```
$("#p").fadeIn("slow",function(){ alert("Animation Done."); });
```

**See Also**

fadeOut(String|Number,Function)  
fadeTo(String|Number,Number,Function)

## fadeOut( speed, callback )

---

Fade out all matched elements by adjusting their opacity and firing an optional callback after completion. Only the opacity is adjusted for this animation, meaning that all of the matched elements should already have some form of height and width associated with them.

**Parameters**

**speed**: *String|Number*: (optional) A string representing one of the three predefined speeds ("slow", "normal", or "fast") or the number of milliseconds to run the animation (e.g. 1000).

**callback**: *Function*: (optional) A function to be executed whenever the animation completes.

**Returns**

jQuery

**Example**

```
$("#p").fadeOut("slow");
```

**Example**

```
$("#p").fadeOut("slow",function(){ alert("Animation Done."); });
```

**See Also**

fadeIn(String|Number,Function)  
fadeTo(String|Number,Number,Function)

## fadeTo( speed, opacity, callback )

---

Fade the opacity of all matched elements to a specified opacity and firing an optional callback after completion. Only the opacity is adjusted for this animation, meaning that all of the matched elements should already have some form of height and width associated with them.

### Parameters

**speed**: *String|Number*: A string representing one of the three predefined speeds ("slow", "normal", or "fast") or the number of milliseconds to run the animation (e.g. 1000).

**opacity**: *Number*: The opacity to fade to (a number from 0 to 1).

**callback**: *Function*: (optional) A function to be executed whenever the animation completes.

### Returns

jQuery

### Example

```
$("#p").fadeTo("slow", 0.5);
```

### Example

```
$("#p").fadeTo("slow", 0.5, function(){ alert("Animation Done."); });
```

### See Also

fadeIn(String|Number,Function)  
fadeOut(String|Number,Function)

## animate( params, speed, easing, callback )

---

A function for making your own, custom, animations. The key aspect of this function is the object of style properties that will be animated, and to what end. Each key within the object represents a style property that will also be animated (for example: "height", "top", or "opacity"). The value associated with the key represents to what end the property will be animated. If a number is provided as the value, then the style property will be transitioned from its current state to that new number. Otherwise if the string "hide", "show", or "toggle" is provided, a default animation will be constructed for that property.

### Parameters

**params**: *Hash*: A set of style attributes that you wish to animate, and to what end.

**speed**: *String|Number*: (optional) A string representing one of the three predefined speeds ("slow", "normal", or "fast") or the number of milliseconds to run the animation (e.g. 1000).

**easing**: *String*: (optional) The name of the easing effect that you want to use (Plugin Required).

**callback**: *Function*: (optional) A function to be executed whenever the animation completes.

### Returns

jQuery

### Example

```
$("#p").animate({ height: 'toggle', opacity: 'toggle' }, "slow");
```

### Example

```
$("#p").animate({ left: 50, opacity: 'show' }, 500);
```

### 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 (Only 'linear' is provided by default, with jQuery).

```
$("#p").animate({ opacity: 'show' }, "slow", "easein");
```

# Events

## bind( type, data, fn )

---

Binds a handler to a particular event (like click) for each matched element. The event handler is passed an event object that you can use to prevent default behaviour. To stop both default action and event bubbling, your handler has to return false. In most cases, you can define your event handlers as anonymous functions (see first example). In cases where that

is not possible, you can pass additional data as the second paramter (and the handler function as the third), see second example.

#### **Parameterss**

**type:** *String*: An event type

**data:** *Object*: (optional) Additional data passed to the event handler as event.data

**fn:** *Function*: A function to bind to the event on each of the set of matched elements

#### **Returns**

jQuery

#### **Example**

```
$("#p").bind("click", function(){ alert( $(this).text() ); });
```

#### **Before**

```
<p>Hello</p>
```

#### **Result**

```
alert("Hello")
```

#### **Example**

Pass some additional data to the event handler.

```
function handler(event) { alert(event.data.foo); } $("#p").bind("click", {foo: "bar"}, handler)
```

#### **Result**

```
alert("bar")
```

#### **Example**

Cancel a default action and prevent it from bubbling by returning false from your function.

```
$("#form").bind("submit", function() { return false; })
```

#### **Example**

Cancel only the default action by using the preventDefault method.

```
$("#form").bind("submit", function(event){ event.preventDefault(); });
```

#### **Example**

Stop only an event from bubbling by using the stopPropagation method.

```
$("#form").bind("submit", function(event){ event.stopPropagation(); });
```

## **one( type, data, fn )**

---

Binds a handler to a particular event (like click) for each matched element. The handler is executed only once for each element. Otherwise, the same rules as described in bind() apply. The event handler is passed an event object that you can use to prevent default behaviour. To stop both default action and event bubbling, your handler has to return false. In most cases, you can define your event handlers as anonymous functions (see first example). In cases where that is not possible, you can pass additional data as the second paramter (and the handler function as the third), see second example.

#### **Parameterss**

**type:** *String*: An event type

**data:** *Object*: (optional) Additional data passed to the event handler as event.data

**fn:** *Function*: A function to bind to the event on each of the set of matched elements

#### **Returns**

jQuery

#### **Example**

```
$("#p").one("click", function(){ alert( $(this).text() ); });
```

#### **Before**

```
<p>Hello</p>
```

#### **Result**

```
alert("Hello")
```

## unbind( type, fn )

---

The opposite of bind, removes a bound event from each of the matched elements. Without any arguments, all bound events are removed. If the type is provided, all bound events of that type are removed. If the function that was passed to bind is provided as the second argument, only that specific event handler is removed.

### Parameters

**type:** *String*: (optional) An event type

**fn:** *Function*: (optional) A function to unbind from the event on each of the set of matched elements

### Returns

jQuery

### Example

```
$("#p").unbind()
```

#### Before

```
<p onclick="alert('Hello');">Hello</p>
```

#### Result

```
[ <p>Hello</p> ]
```

### Example

```
$("#p").unbind( "click" )
```

#### Before

```
<p onclick="alert('Hello');">Hello</p>
```

#### Result

```
[ <p>Hello</p> ]
```

### Example

```
$("#p").unbind( "click", function() { alert("Hello"); } )
```

#### Before

```
<p onclick="alert('Hello');">Hello</p>
```

#### Result

```
[ <p>Hello</p> ]
```

## trigger( type )

---

Trigger a type of event on every matched element.

### Parameters

**type:** *String*: An event type to trigger.

### Returns

jQuery

### Example

```
$("#p").trigger("click")
```

#### Before

```
<p click="alert('hello')">Hello</p>
```

#### Result

```
alert('hello')
```

## toggle( even, odd )

---

Toggle between two function calls every other click. Whenever a matched element is clicked, the first specified function is fired, when clicked again, the second is fired. All subsequent clicks continue to rotate through the two functions. Use unbind("click") to remove.

### Parameters

**even**: *Function*: The function to execute on every even click.

**odd**: *Function*: The function to execute on every odd click.

**Returns**

jQuery

**Example**

```
$("#p").toggle(function(){ $(this).addClass("selected"); },function(){ $(this).removeClass("selected"); });
```

---

## hover( over, out )

A method for simulating hovering (moving the mouse on, and off, an object). This is a custom method which provides an 'in' to a frequent task. Whenever the mouse cursor is moved over a matched element, the first specified function is fired. Whenever the mouse moves off of the element, the second specified function fires. Additionally, checks are in place to see if the mouse is still within the specified element itself (for example, an image inside of a div), and if it is, it will continue to 'hover', and not move out (a common error in using a mouseout event handler).

**Parameters**

**over**: *Function*: The function to fire whenever the mouse is moved over a matched element.

**out**: *Function*: The function to fire whenever the mouse is moved off of a matched element.

**Returns**

jQuery

**Example**

```
$("#p").hover(function(){ $(this).addClass("over"); },function(){ $(this).addClass("out"); });
```

---

## ready( fn )

Bind a function to be executed whenever the DOM is ready to be traversed and manipulated. This is probably the most important function included in the event module, as it can greatly improve the response times of your web applications. In a nutshell, this is a solid replacement for using window.onload, and attaching a function to that. By using this method, your bound Function will be called the instant the DOM is ready to be read and manipulated, which is exactly what 99.99% of all Javascript code needs to run. Please ensure you have no code in your <body> onload event handler, otherwise \$(document).ready() may not fire. You can have as many \$(document).ready events on your page as you like. The functions are then executed in the order they were added.

**Parameters**

**fn**: *Function*: The function to be executed when the DOM is ready.

**Returns**

jQuery

**Example**

```
$(document).ready(function(){ Your code here... });
```

---

## scroll( fn )

Bind a function to the scroll event of each matched element.

**Parameters**

**fn**: *Function*: A function to bind to the scroll event on each of the matched elements.

**Returns**

jQuery

**Example**

```
$("#p").scroll( function() { alert("Hello"); } );
```

**Before**

```
<p>Hello</p>
```

**Result**

```
<p onscroll="alert('Hello');">Hello</p>
```

## submit( fn )

---

Bind a function to the submit event of each matched element.

### Parameters

**fn**: *Function*: A function to bind to the submit event on each of the matched elements.

### Returns

jQuery

### Example

Prevents the form submission when the input has no value entered.

```
$("#myform").submit( function() { return $("input", this).val().length > 0; } );
```

### Before

```
<form id="myform"><input /></form>
```

## submit( )

---

Trigger the submit event of each matched element. This causes all of the functions that have been bound to that submit event to be executed. Note: This does not execute the submit method of the form element! If you need to submit the form via code, you have to use the DOM method, eg. `$("#form")[0].submit()`;

### Returns

jQuery

### Example

Triggers all submit events registered for forms, but does not submit the form

```
$("#form").submit();
```

## focus( fn )

---

Bind a function to the focus event of each matched element.

### Parameters

**fn**: *Function*: A function to bind to the focus event on each of the matched elements.

### Returns

jQuery

### Example

```
$("#p").focus( function() { alert("Hello"); } );
```

### Before

```
<p>Hello</p>
```

### Result

```
<p onfocus="alert('Hello');">Hello</p>
```

## focus( )

---

Trigger the focus event of each matched element. This causes all of the functions that have been bound to that focus event to be executed. Note: This does not execute the focus method of the underlying elements! If you need to focus an element via code, you have to use the DOM method, eg. `$("#myinput")[0].focus()`;

### Returns

jQuery

### Example

```
$("#p").focus();
```

### Before

```
<p onfocus="alert('Hello');">Hello</p>
```

**Result**

```
alert('Hello');
```

## keydown( fn )

---

Bind a function to the keydown event of each matched element.

**Parameters**

**fn**: *Function*: A function to bind to the keydown event on each of the matched elements.

**Returns**

jQuery

**Example**

```
$("p").keydown( function() { alert("Hello"); } );
```

**Before**

```
<p>Hello</p>
```

**Result**

```
<p onkeydown="alert('Hello');">Hello</p>
```

## dblclick( fn )

---

Bind a function to the dblclick event of each matched element.

**Parameters**

**fn**: *Function*: A function to bind to the dblclick event on each of the matched elements.

**Returns**

jQuery

**Example**

```
$("p").dblclick( function() { alert("Hello"); } );
```

**Before**

```
<p>Hello</p>
```

**Result**

```
<p ondblclick="alert('Hello');">Hello</p>
```

## keypress( fn )

---

Bind a function to the keypress event of each matched element.

**Parameters**

**fn**: *Function*: A function to bind to the keypress event on each of the matched elements.

**Returns**

jQuery

**Example**

```
$("p").keypress( function() { alert("Hello"); } );
```

**Before**

```
<p>Hello</p>
```

**Result**

```
<p onkeypress="alert('Hello');">Hello</p>
```

## error( fn )

---

Bind a function to the error event of each matched element.

**Parameters**

**fn:** *Function:* A function to bind to the error event on each of the matched elements.

**Returns**

jQuery

**Example**

```
$("#p").error( function() { alert("Hello"); } );
```

**Before**

```
<p>Hello</p>
```

**Result**

```
<p onerror="alert('Hello');">Hello</p>
```

---

## blur( fn )

Bind a function to the blur event of each matched element.

**Parameters**

**fn:** *Function:* A function to bind to the blur event on each of the matched elements.

**Returns**

jQuery

**Example**

```
$("#p").blur( function() { alert("Hello"); } );
```

**Before**

```
<p>Hello</p>
```

**Result**

```
<p onblur="alert('Hello');">Hello</p>
```

---

## blur( )

Trigger the blur event of each matched element. This causes all of the functions that have been bound to that blur event to be executed. Note: This does not execute the blur method of the underlying elements! If you need to blur an element via code, you have to use the DOM method, eg. `$("#myinput")[0].blur();`

**Returns**

jQuery

**Example**

```
$("#p").blur();
```

**Before**

```
<p onblur="alert('Hello');">Hello</p>
```

**Result**

```
alert('Hello');
```

---

## load( fn )

Bind a function to the load event of each matched element.

**Parameters**

**fn:** *Function:* A function to bind to the load event on each of the matched elements.

**Returns**

jQuery

**Example**

```
$("#p").load( function() { alert("Hello"); } );
```

**Before**

```
<p>Hello</p>
```

**Result**

```
<p onload="alert('Hello');">Hello</p>
```

## select( fn )

---

Bind a function to the select event of each matched element.

**Parameters**

**fn**: *Function*: A function to bind to the select event on each of the matched elements.

**Returns**

jQuery

**Example**

```
$("#p").select( function() { alert("Hello"); } );
```

**Before**

```
<p>Hello</p>
```

**Result**

```
<p onselect="alert('Hello');">Hello</p>
```

## select( )

---

Trigger the select event of each matched element. This causes all of the functions that have been bound to that select event to be executed.

**Returns**

jQuery

**Example**

```
$("#p").select();
```

**Before**

```
<p onselect="alert('Hello');">Hello</p>
```

**Result**

```
alert('Hello');
```

## mouseup( fn )

---

Bind a function to the mouseup event of each matched element.

**Parameters**

**fn**: *Function*: A function to bind to the mouseup event on each of the matched elements.

**Returns**

jQuery

**Example**

```
$("#p").mouseup( function() { alert("Hello"); } );
```

**Before**

```
<p>Hello</p>
```

**Result**

```
<p onmouseup="alert('Hello');">Hello</p>
```

## unload( fn )

---

Bind a function to the unload event of each matched element.

### Parameters

**fn**: *Function*: A function to bind to the unload event on each of the matched elements.

### Returns

jQuery

### Example

```
$("#p").unload( function() { alert("Hello"); } );
```

### Before

```
<p>Hello</p>
```

### Result

```
<p onunload="alert('Hello');">Hello</p>
```

## change( fn )

---

Bind a function to the change event of each matched element.

### Parameters

**fn**: *Function*: A function to bind to the change event on each of the matched elements.

### Returns

jQuery

### Example

```
$("#p").change( function() { alert("Hello"); } );
```

### Before

```
<p>Hello</p>
```

### Result

```
<p onchange="alert('Hello');">Hello</p>
```

## mouseout( fn )

---

Bind a function to the mouseout event of each matched element.

### Parameters

**fn**: *Function*: A function to bind to the mouseout event on each of the matched elements.

### Returns

jQuery

### Example

```
$("#p").mouseout( function() { alert("Hello"); } );
```

### Before

```
<p>Hello</p>
```

### Result

```
<p onmouseout="alert('Hello');">Hello</p>
```

## keyup( fn )

---

Bind a function to the keyup event of each matched element.

### Parameters

**fn**: *Function*: A function to bind to the keyup event on each of the matched elements.

### Returns

jQuery

### **Example**

```
$("#p").keyup( function() { alert("Hello"); } );
```

### **Before**

```
<p>Hello</p>
```

### **Result**

```
<p onkeyup="alert('Hello');">Hello</p>
```

## **click( fn )**

---

Bind a function to the click event of each matched element.

### **Parameters**

**fn**: *Function*: A function to bind to the click event on each of the matched elements.

### **Returns**

jQuery

### **Example**

```
$("#p").click( function() { alert("Hello"); } );
```

### **Before**

```
<p>Hello</p>
```

### **Result**

```
<p onclick="alert('Hello');">Hello</p>
```

## **click( )**

---

Trigger the click event of each matched element. This causes all of the functions that have been bound to that click event to be executed.

### **Returns**

jQuery

### **Example**

```
$("#p").click();
```

### **Before**

```
<p onclick="alert('Hello');">Hello</p>
```

### **Result**

```
alert('Hello');
```

## **resize( fn )**

---

Bind a function to the resize event of each matched element.

### **Parameters**

**fn**: *Function*: A function to bind to the resize event on each of the matched elements.

### **Returns**

jQuery

### **Example**

```
$("#p").resize( function() { alert("Hello"); } );
```

### **Before**

```
<p>Hello</p>
```

### **Result**

```
<p onresize="alert('Hello');">Hello</p>
```

## mousemove( fn )

---

Bind a function to the mousemove event of each matched element.

### Parameters

**fn:** *Function:* A function to bind to the mousemove event on each of the matched elements.

### Returns

jQuery

### Example

```
$("#p").mousemove( function() { alert("Hello"); } );
```

### Before

```
<p>Hello</p>
```

### Result

```
<p onmousemove="alert('Hello');">Hello</p>
```

## mousedown( fn )

---

Bind a function to the mousedown event of each matched element.

### Parameters

**fn:** *Function:* A function to bind to the mousedown event on each of the matched elements.

### Returns

jQuery

### Example

```
$("#p").mousedown( function() { alert("Hello"); } );
```

### Before

```
<p>Hello</p>
```

### Result

```
<p onmousedown="alert('Hello');">Hello</p>
```

## mouseover( fn )

---

Bind a function to the mouseover event of each matched element.

### Parameters

**fn:** *Function:* A function to bind to the mouseover event on each of the matched elements.

### Returns

jQuery

### Example

```
$("#p").mouseover( function() { alert("Hello"); } );
```

### Before

```
<p>Hello</p>
```

### Result

```
<p onmouseover="alert('Hello');">Hello</p>
```

## loadIfModified( url, params, callback )

---

Load HTML from a remote file and inject it into the DOM, only if it's been modified by the server.

### Parameters

**url:** *String*: The URL of the HTML file to load.

**params:** *Map*: (optional) Key/value pairs that will be sent to the server.

**callback:** *Function*: (optional) A function to be executed whenever the data is loaded (parameters: responseText, status and response itself).

### Returns

jQuery

### Example

```
$("#feeds").loadIfModified("feeds.html");
```

#### Before

```
<div id="feeds"></div>
```

#### Result

```
<div id="feeds"><b>45</b> feeds found.</div>
```

## load( url, params, callback )

---

Load HTML from a remote file and inject it into the DOM. Note: Avoid to use this to load scripts, instead use `$.getScript`. IE strips script tags when there aren't any other characters in front of it.

### Parameters

**url:** *String*: The URL of the HTML file to load.

**params:** *Object*: (optional) A set of key/value pairs that will be sent as data to the server.

**callback:** *Function*: (optional) A function to be executed whenever the data is loaded (parameters: responseText, status and response itself).

### Returns

jQuery

### Example

```
$("#feeds").load("feeds.html");
```

#### Before

```
<div id="feeds"></div>
```

#### Result

```
<div id="feeds"><b>45</b> feeds found.</div>
```

### Example

Same as above, but with an additional parameter and a callback that is executed when the data was loaded.

```
$("#feeds").load("feeds.html", {limit: 25}, function() { alert("The last 25 entries in the feed have been loaded"); } );
```

## serialize( )

---

Serializes a set of input elements into a string of data. This will serialize all given elements. A serialization similar to the form submit of a browser is provided by the form plugin. It also takes multiple-selects into account, while this method recognizes only a single option.

### Returns

String

### Example

Serialize a selection of input elements to a string

```
$("#input[@type=text]").serialize();
```

#### Before

```
<input type='text' name='name' value='John'/> <input type='text' name='location' value='Boston'/>
```

## evalScripts( )

---

Evaluate all script tags inside this jQuery. If they have a src attribute, the script is loaded, otherwise it's content is evaluated.

### Returns

jQuery

## ajaxStart( callback )

---

Attach a function to be executed whenever an AJAX request begins and there is none already active.

### Parameters

**callback:** *Function:* The function to execute.

### Returns

jQuery

### Example

Show a loading message whenever an AJAX request starts (and none is already active).

```
$("#loading").ajaxStart(function(){ $(this).show(); });
```

## ajaxStop( callback )

---

Attach a function to be executed whenever all AJAX requests have ended.

### Parameters

**callback:** *Function:* The function to execute.

### Returns

jQuery

### Example

Hide a loading message after all the AJAX requests have stopped.

```
$("#loading").ajaxStop(function(){ $(this).hide(); });
```

## ajaxComplete( callback )

---

Attach a function to be executed whenever an AJAX request completes. The XMLHttpRequest and settings used for that request are passed as arguments to the callback.

### Parameters

**callback:** *Function:* The function to execute.

### Returns

jQuery

### Example

Show a message when an AJAX request completes.

```
$("#msg").ajaxComplete(function(request, settings){ $(this).append("<li>Request Complete.</li>"); });
```

## ajaxSuccess( callback )

---

Attach a function to be executed whenever an AJAX request completes successfully. The XMLHttpRequest and settings used for that request are passed as arguments to the callback.

### Parameters

**callback:** *Function*: The function to execute.

**Returns**

jQuery

**Example**

Show a message when an AJAX request completes successfully.

```
$("#msg").ajaxSuccess(function(request, settings){ $(this).append("<li>Successful Request!</li>"); });
```

---

## ajaxError( callback )

Attach a function to be executed whenever an AJAX request fails. The XMLHttpRequest and settings used for that request are passed as arguments to the callback. A third argument, an exception object, is passed if an exception occurred while processing the request.

**Parameters**

**callback:** *Function*: The function to execute.

**Returns**

jQuery

**Example**

Show a message when an AJAX request fails.

```
$("#msg").ajaxError(function(request, settings){ $(this).append("<li>Error requesting page " + settings.url + "</li>"); });
```

---

## ajaxSend( callback )

Attach a function to be executed before an AJAX request is send. The XMLHttpRequest and settings used for that request are passed as arguments to the callback.

**Parameters**

**callback:** *Function*: The function to execute.

**Returns**

jQuery

**Example**

Show a message before an AJAX request is send.

```
$("#msg").ajaxSend(function(request, settings){ $(this).append("<li>Starting request at " + settings.url + "</li>"); });
```

---

## \$.get( url, params, callback )

Load a remote page using an HTTP GET request.

**Parameters**

**url:** *String*: The URL of the page to load.

**params:** *Map*: (optional) Key/value pairs that will be sent to the server.

**callback:** *Function*: (optional) A function to be executed whenever the data is loaded.

**Returns**

XMLHttpRequest

**Example**

```
$.get("test.cgi");
```

**Example**

```
$.get("test.cgi", { name: "John", time: "2pm" });
```

**Example**

```
$.get("test.cgi", function(data){ alert("Data Loaded: " + data); });
```

**Example**

```
$.get("test.cgi", { name: "John", time: "2pm" }, function(data){ alert("Data Loaded: " + data); } );
```

## \$.getIfModified( url, params, callback )

---

Load a remote page using an HTTP GET request, only if it hasn't been modified since it was last retrieved.

### Parameters

**url:** *String*: The URL of the page to load.

**params:** *Map*: (optional) Key/value pairs that will be sent to the server.

**callback:** *Function*: (optional) A function to be executed whenever the data is loaded.

### Returns

XMLHttpRequest

### Example

```
$.getIfModified("test.html");
```

### Example

```
$.getIfModified("test.html", { name: "John", time: "2pm" } );
```

### Example

```
$.getIfModified("test.cgi", function(data){ alert("Data Loaded: " + data); } );
```

### Example

```
$.getIfModified("test.cgi", { name: "John", time: "2pm" }, function(data){ alert("Data Loaded: " + data); } );
```

## \$.getScript( url, callback )

---

Loads, and executes, a remote JavaScript file using an HTTP GET request. Warning: Safari <= 2.0.x is unable to evaluate scripts in a global context synchronously. If you load functions via getScript, make sure to call them after a delay.

### Parameters

**url:** *String*: The URL of the page to load.

**callback:** *Function*: (optional) A function to be executed whenever the data is loaded.

### Returns

XMLHttpRequest

### Example

```
$.getScript("test.js");
```

### Example

```
$.getScript("test.js", function(){ alert("Script loaded and executed."); } );
```

## \$.getJSON( url, params, callback )

---

Load JSON data using an HTTP GET request.

### Parameters

**url:** *String*: The URL of the page to load.

**params:** *Map*: (optional) Key/value pairs that will be sent to the server.

**callback:** *Function*: A function to be executed whenever the data is loaded.

### Returns

XMLHttpRequest

### Example

```
$.getJSON("test.js", function(json){ alert("JSON Data: " + json.users[3].name); } );
```

### Example

```
$.getJSON("test.js", { name: "John", time: "2pm" }, function(json){ alert("JSON Data: " + json.users[3].name); } );
```

## **\$.post( url, params, callback )**

---

Load a remote page using an HTTP POST request.

### **Parameters**

**url:** *String:* The URL of the page to load.

**params:** *Map:* (optional) Key/value pairs that will be sent to the server.

**callback:** *Function:* (optional) A function to be executed whenever the data is loaded.

### **Returns**

XMLHttpRequest

### **Example**

```
$.post("test.cgi");
```

### **Example**

```
$.post("test.cgi", { name: "John", time: "2pm" } );
```

### **Example**

```
$.post("test.cgi", function(data){ alert("Data Loaded: " + data); });
```

### **Example**

```
$.post("test.cgi", { name: "John", time: "2pm" }, function(data){ alert("Data Loaded: " + data); } );
```

## **\$.ajaxTimeout( time )**

---

Set the timeout of all AJAX requests to a specific amount of time. This will make all future AJAX requests timeout after a specified amount of time. Set to null or 0 to disable timeouts (default). You can manually abort requests with the XMLHttpRequest's (returned by all ajax functions) abort() method. Deprecated. Use \$.ajaxSetup instead.

### **Parameters**

**time:** *Number:* How long before an AJAX request times out.

### **Returns**

undefined

### **Example**

Make all AJAX requests timeout after 5 seconds.

```
$.ajaxTimeout( 5000 );
```

## **\$.ajaxSetup( settings )**

---

Setup global settings for AJAX requests. See \$.ajax for a description of all available options.

### **Parameters**

**settings:** *Map:* Key/value pairs to use for all AJAX requests

### **Returns**

undefined

### **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.

```
$.ajaxSetup( { url: "/xmlhttp/", global: false, type: "POST" } ); $.ajax({ data: myData });
```

## **\$.ajax( properties )**

---

Load a remote page using an HTTP request. This is jQuery's low-level AJAX implementation. See \$.get, \$.post etc. for higher-level abstractions. \$.ajax() returns the XMLHttpRequest that it creates. In most cases you won't need that object to manipulate directly, but it is available if you need to abort the request manually. Note: Make sure the server sends the right mimetype (eg. xml as "text/xml"). Sending the wrong mimetype will get you into serious trouble that jQuery can't

solve. Supported datatypes are (see dataType option): "xml": Returns a XML document that can be processed via jQuery. "html": Returns HTML as plain text, included script tags are evaluated. "script": Evaluates the response as Javascript and returns it as plain text. "json": Evaluates the response as JSON and returns a Javascript Object \$.ajax() takes one argument, an object of key/value pairs, that are used to initialize and handle the request. These are all the key/values that can be used: (String) url - The URL to request. (String) type - The type of request to make ("POST" or "GET"), default is "GET". (String) dataType - The type of data that you're expecting back from the server. No default: If the server sends xml, the responseXML, otherwise the responseText is passed to the success callback. (Boolean) ifModified - 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. (Number) timeout - Local timeout to override global timeout, eg. to give a single request a longer timeout while all others timeout after 1 second. See \$.ajaxTimeout() for global timeouts. (Boolean) global - Whether to trigger global AJAX event handlers for this request, default is true. Set to false to prevent that global handlers like ajaxStart or ajaxStop are triggered. (Function) error - A function to be called if the request fails. The function gets passed three arguments: The XMLHttpRequest object, a string describing the type of error that occurred and an optional exception object, if one occurred. (Function) success - A function to be called if the request succeeds. The function gets passed one argument: The data returned from the server, formatted according to the 'dataType' parameter. (Function) complete - A function to be called when the request finishes. The function gets passed two arguments: The XMLHttpRequest object and a string describing the type of success of the request. (Object|String) data - Data to be sent to the server. Converted to a query string, if not already a string. Is appended to the url for GET-requests. See processData option to prevent this automatic processing. (String) contentType - When sending data to the server, use this content-type. Default is "application/x-www-form-urlencoded", which is fine for most cases. (Boolean) processData - By default, data passed in to the data option as an object other as 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 DOMDocuments, set this option to false. (Boolean) async - By default, all requests are send asynchronous (set to true). If you need synchronous requests, set this option to false. (Function) beforeSend - A pre-callback to set custom headers etc., the XMLHttpRequest is passed as the only argument.

#### Parameters

**properties:** Map: Key/value pairs to initialize the request with.

#### Returns

XMLHttpRequest

#### Example

Load and execute a JavaScript file.

```
$.ajax({ type: "GET", url: "test.js", dataType: "script" })
```

#### Example

Save some data to the server and notify the user once its complete.

```
$.ajax({ type: "POST", url: "some.php", data: "name=John&location=Boston", success: function(msg){ alert( "Data Saved: " + msg ); } });
```

#### Example

Loads data synchronously. Blocks the browser while the requests is active. It is better to block user interaction with others means when synchronization is necessary, instead to block the complete browser.

```
var html = $.ajax({ url: "some.php", async: false }).responseText;
```

#### Example

Sends an xml document as data to the server. By setting the processData option to false, the automatic conversion of data to strings is prevented.

```
var xmlDocument = [create xml document]; $.ajax({ url: "page.php", processData: false, data: xmlDocument, success: handleResponse });
```

## Plugins

### Button

#### button( hash )

Creates a button from an image element. This function attempts to mimic the functionality of the "button" found in modern day GUIs. There are two different buttons you can create using this plugin; Normal buttons, and Toggle buttons.

#### Parameters

**hash:** *hOptions*: with options, described below. *sPath* Full path to the images, either relative or with full URL *sExt* Extension of the used images (jpg|gif|png) *sName* Name of the button, if not specified, try to fetch from *id* *iWidth* Width of the button, if not specified, try to fetch from element.*width* *iHeight* Height of the button, if not specified, try to fetch from element.*height* *onAction* Function to call when clicked / toggled. In case of a string, the element is wrapped inside an href tag. *bToggle* Do we need to create a togglebutton? (boolean) *bState* Initial state of the button? (boolean) *sType* Type of hover to create (img|css)

#### Returns

## Center

---

### center( )

---

Takes all matched elements and centers them, absolutely, within the context of their parent element. Great for doing slideshows.

**Returns**

jQuery

**Example**

```
$("#div img").center();
```

## Cookie

---

### \$.cookie( name, value, options )

---

Create a cookie with the given name and value and other optional parameters.

**Parameters**

**name:** *String*: The name of the cookie.

**value:** *String*: The value of the cookie.

**options:** *Object*: An object literal containing key/value pairs to provide optional cookie attributes.

**Hash Options**

**expires:** *Number|Date*: Either an integer specifying the expiration date from now on in days or a Date object. If a negative value is specified (e.g. a date in the past), the cookie will be deleted. If set to null or omitted, the cookie will be a session cookie and will not be retained when the browser exits.

**path:** *String*: The value of the path attribute of the cookie (default: path of page that created the cookie).

**domain:** *String*: The value of the domain attribute of the cookie (default: domain of page that created the cookie).

**secure:** *Boolean*: If true, the secure attribute of the cookie will be set and the cookie transmission will require a secure protocol (like HTTPS).

**Returns**

undefined

**Example**

Set the value of a cookie.

```
$.cookie('the_cookie', 'the_value');
```

**Example**

Create a cookie with all available options.

```
$.cookie('the_cookie', 'the_value', {expires: 7, path: '/', domain: 'jquery.com', secure: true});
```

**Example**

Create a session cookie.

```
$.cookie('the_cookie', 'the_value');
```

**Example**

Delete a cookie by setting a date in the past.

```
$.cookie('the_cookie', '', {expires: -1});
```

### \$.cookie( name )

---

Get the value of a cookie with the given name.

### **Parameters**

**name:** *String*: The name of the cookie.

### **Returns**

String

### **Example**

Get the value of a cookie.

```
$.cookie('the_cookie');
```

## **Dimensions**

---

### **height( )**

---

Returns the css height value for the first matched element. If used on document, returns the document's height (innerHeight) If used on window, returns the viewport's (window) height

#### **Returns**

Object

#### **Example**

```
$("#testdiv").height()
```

#### **Result**

"200px"

#### **Example**

```
$(document).height();
```

#### **Result**

800

#### **Example**

```
$(window).height();
```

#### **Result**

400

### **width( )**

---

Returns the css width value for the first matched element. If used on document, returns the document's width (innerWidth) If used on window, returns the viewport's (window) width

#### **Returns**

Object

#### **Example**

```
$("#testdiv").width()
```

#### **Result**

"200px"

#### **Example**

```
$(document).width();
```

#### **Result**

800

#### **Example**

```
$(window).width();
```

#### **Result**

400

### **innerHeight( )**

---

Returns the inner height value (without border) for the first matched element. If used on document, returns the document's height (innerHeight) If used on window, returns the viewport's (window) height

**Returns**

Number

**Example**

```
$("#testdiv").innerHeight()
```

**Result**

800

---

## innerWidth( )

Returns the inner width value (without border) for the first matched element. If used on document, returns the document's Width (innerWidth) If used on window, returns the viewport's (window) width

**Returns**

Number

**Example**

```
$("#testdiv").innerWidth()
```

**Result**

1000

---

## outerHeight( )

Returns the outer height value (including border) for the first matched element. Cannot be used on document or window.

**Returns**

Number

**Example**

```
$("#testdiv").outerHeight()
```

**Result**

1000

---

## outerWidth( )

Returns the outer width value (including border) for the first matched element. Cannot be used on document or window.

**Returns**

Number

**Example**

```
$("#testdiv").outerWidth()
```

**Result**

1000

---

## scrollLeft( )

Returns how many pixels the user has scrolled to the right (scrollLeft). Works on containers with overflow: auto and window/document.

**Returns**

Number

**Example**

```
$("#testdiv").scrollLeft()
```

**Result**

100

## scrollTop( )

---

Returns how many pixels the user has scrolled to the bottom (scrollTop). Works on containers with overflow: auto and window/document.

**Returns**

Number

**Example**

```
$("#testdiv").scrollTop()
```

**Result**

100

## offset( )

---

This returns an object with top, left, width, height, borderLeft, borderTop, marginLeft, marginTop, scrollLeft, scrollTop, pageXOffset, pageYOffset. The top and left values include the scroll offsets but the scrollLeft and scrollTop properties of the returned object are the combined scroll offsets of the parent elements (not including the window scroll offsets). This is not the same as the element's scrollTop and scrollLeft. For accurate readings make sure to use pixel values.

**Returns**

Object

## offset( refElement )

---

This returns an object with top, left, width, height, borderLeft, borderTop, marginLeft, marginTop, scrollLeft, scrollTop, pageXOffset, pageYOffset. The top and left values include the scroll offsets but the scrollLeft and scrollTop properties of the returned object are the combined scroll offsets of the parent elements (not including the window scroll offsets). This is not the same as the element's scrollTop and scrollLeft. For accurate readings make sure to use pixel values.

**Parameters**

**refElement**: *String*: This is an expression. The offset returned will be relative to the first matched element.

**Returns**

Object

## offset( refElement )

---

This returns an object with top, left, width, height, borderLeft, borderTop, marginLeft, marginTop, scrollLeft, scrollTop, pageXOffset, pageYOffset. The top and left values include the scroll offsets but the scrollLeft and scrollTop properties of the returned object are the combined scroll offsets of the parent elements (not including the window scroll offsets). This is not the same as the element's scrollTop and scrollLeft. For accurate readings make sure to use pixel values.

**Parameters**

**refElement**: *jQuery*: The offset returned will be relative to the first matched element.

**Returns**

Object

## offset( refElement )

---

This returns an object with top, left, width, height, borderLeft, borderTop, marginLeft, marginTop, scrollLeft, scrollTop, pageXOffset, pageYOffset. The top and left values include the scroll offsets but the scrollLeft and scrollTop properties of the returned object are the combined scroll offsets of the parent elements (not including the window scroll offsets). This is not the same as the element's scrollTop and scrollLeft. For accurate readings make sure to use pixel values.

**Parameters**

**refElement:** *HTMLElement*: The offset returned will be relative to this element.

**Returns**

Object

## Metadata

---

### **\$.meta.setType( type, name )**

---

Sets the type of metadata to use. Metadata is encoded in JSON, and each property in the JSON will become a property of the element itself. There are three supported types of metadata storage: **attr**: Inside an attribute. The name parameter indicates *which* attribute. **class**: Inside the class attribute, wrapped in curly braces: { } **elem**: Inside a child element (e.g. a script tag). The name parameter indicates *which* element. The metadata for an element is loaded the first time the element is accessed via jQuery. As a result, you can define the metadata type, use `$(expr)` to load the metadata into the elements matched by `expr`, then redefine the metadata type and run another `$(expr)` for other elements.

**Parameters**

**type:** *String*: The encoding type

**name:** *String*: The name of the attribute to be used to get metadata (optional)

**Returns**

undefined

**Example**

Reads metadata from the class attribute

```
<p id="one" class="some_class {item_id: 1, item_label: 'Label'}">This is a p</p>
```

**Before**

```
$.meta.setType("class")
```

**Example**

Reads metadata from a "data" attribute

```
<p id="one" class="some_class" data="{item_id: 1, item_label: 'Label'}">This is a p</p>
```

**Before**

```
$.meta.setType("attr", "data")
```

**Example**

Reads metadata from a nested script element

```
<p id="one" class="some_class"><script>{item_id: 1, item_label: 'Label'}</script>This is a p</p>
```

**Before**

```
$.meta.setType("elem", "script")
```

### **data( )**

---

Returns the metadata object for the first member of the jQuery object.

**Returns**

jQuery

## Form

---

### **ajaxSubmit( object )**

---

`ajaxSubmit()` provides a mechanism for submitting an HTML form using AJAX. `ajaxSubmit` accepts a single argument which can be either a success callback function or an options Object. If a function is provided it will be invoked upon successful completion of the submit and will be passed the response from the server. If an options Object is provided, the following attributes are supported: **target**: Identifies the element(s) in the page to be updated with the server response. This value may be specified as a jQuery selection string, a jQuery object, or a DOM element. **default value**: null **url**: URL to which the form data will be submitted. **default value**: value of form's 'action' attribute method:

**Parameters**

**object:** *options*: literal containing options which control the form submission process

### Returns

jQuery

### Example

Submit form and alert server response

```
$('#myForm').ajaxSubmit(function(data) { alert('Form submit succeeded! Server returned: ' + data); });
```

### Example

Submit form and update page element with server response

```
var options = { target: '#myTargetDiv' }; $('#myForm').ajaxSubmit(options);
```

### Example

Submit form and alert the server response

```
var options = { success: function(responseText) { alert(responseText); } }; $('#myForm').ajaxSubmit(options);
```

### Example

Pre-submit validation which aborts the submit operation if form data is empty

```
var options = { beforeSubmit: function(formArray, jqForm) { if (formArray.length == 0) { alert('Please enter data.');
```

```
return false; } } }; $('#myForm').ajaxSubmit(options);
```

### Example

json data returned and evaluated

```
var options = { url: myJsonUrl.php, dataType: 'json', success: function(data) { // 'data' is an object representing the the  
evaluated json data } }; $('#myForm').ajaxSubmit(options);
```

### Example

XML data returned from server

```
var options = { url: myXmlUrl.php, dataType: 'xml', success: function(responseXML) { // responseXML is XML document  
object var data = $('myElement', responseXML).text(); } }; $('#myForm').ajaxSubmit(options);
```

### Example

submit form and reset it if successful

```
var options = { resetForm: true }; $('#myForm').ajaxSubmit(options);
```

### Example

Bind form's submit event to use ajaxSubmit

```
$('#myForm').submit(function() { $(this).ajaxSubmit(); return false; });
```

### See Also

formToArray  
ajaxForm  
\$.ajax

## ajaxForm( object )

---

ajaxForm() provides a mechanism for fully automating form submission. The advantages of using this method instead of ajaxSubmit() are: 1. This method will include coordinates for <input type="image" /> elements (if the element is used to submit the form). 2. This method will include the submit element's name/value data (for the element that was used to submit the form). 3. This method binds the submit() method to the form for you. Note that for accurate x/y coordinates of image submit elements in all browsers you need to also use the "dimensions" plugin (this method will auto-detect its presence). The options argument for ajaxForm works exactly as it does for ajaxSubmit. ajaxForm merely passes the options argument along after properly binding events for submit elements and the form itself. See ajaxSubmit for a full description of the options argument.

### Parameters

**object:** *options*: literal containing options which control the form submission process

### Returns

jQuery

### Example

Bind form's submit event so that 'myTargetDiv' is updated with the server response when the form is submitted.

```
var options = { target: '#myTargetDiv' }; $('#myForm').ajaxSForm(options);
```

### Example

Bind form's submit event so that server response is alerted after the form is submitted.

```
var options = { success: function(responseText) { alert(responseText); } }; $('#myForm').ajaxSubmit(options);
```

#### **Example**

Bind form's submit event so that pre-submit callback is invoked before the form is submitted.

```
var options = { beforeSubmit: function(formArray, jqForm) { if (formArray.length == 0) { alert('Please enter data.');
```

```
return false; } } }; $('#myForm').ajaxSubmit(options);
```

## **formToArray( true )**

---

formToArray() gathers form element data into an array of objects that can be passed to any of the following ajax functions: \$.get, \$.post, or load. Each object in the array has both a 'name' and 'value' property. An example of an array for a simple login form might be: [ { name: 'username', value: 'jresig' }, { name: 'password', value: 'secret' } ] It is this array that is passed to pre-submit callback functions provided to the ajaxSubmit() and ajaxForm() methods. The semantic argument can be used to force form serialization in semantic order. If your form must be submitted with name/value pairs in semantic order then pass true for this arg, otherwise pass false (or nothing) to avoid the overhead for this logic (which can be significant for very large forms).

#### **Parameters**

**true**: *semantic*: if serialization must maintain strict semantic ordering of elements (slower)

#### **Returns**

Array<Object>

#### **Example**

Collect all the data from a form and submit it to the server.

```
var data = $("#myForm").formToArray(); $.post( "myscript.cgi", data );
```

#### **See Also**

ajaxForm  
ajaxSubmit

## **formSerialize( true )**

---

Serializes form data into a 'submittable' string. This method will return a string in the format: name1=value1&name2=value2 The semantic argument can be used to force form serialization in semantic order. If your form must be submitted with name/value pairs in semantic order then pass true for this arg, otherwise pass false (or nothing) to avoid the overhead for this logic (which can be significant for very large forms).

#### **Parameters**

**true**: *semantic*: if serialization must maintain strict semantic ordering of elements (slower)

#### **Returns**

String

#### **Example**

Collect all the data from a form into a single string

```
var data = $("#myForm").formSerialize(); $.ajax('POST', "myscript.cgi", data);
```

## **fieldSerialize( true )**

---

Serializes all field elements in the jQuery object into a query string. This method will return a string in the format: name1=value1&name2=value2 The successful argument controls whether or not serialization is limited to 'successful' controls (per <http://www.w3.org/TR/html4/interact/forms.html#successful-controls>). The default value of the successful argument is true.

#### **Parameters**

**true**: *successful*: if only successful controls should be serialized (default is true)

#### **Returns**

String

#### **Example**

Collect the data from all successful input elements into a query string

```
var data = $("input").formSerialize();
```

#### **Example**

Collect the data from all successful radio input elements into a query string

```
var data = $(".radio").formSerialize();
```

#### **Example**

Collect the data from all successful checkbox input elements in myForm into a query string

```
var data = $("#myForm :checkbox").formSerialize();
```

#### **Example**

Collect the data from all checkbox elements in myForm (even the unchecked ones) into a query string

```
var data = $("#myForm :checkbox").formSerialize(false);
```

#### **Example**

Collect the data from all successful input, select, textarea and button elements into a query string

```
var data = $(".input").formSerialize();
```

## **fieldValue( successful )**

---

Returns the value of the field element in the jQuery object. If there is more than one field element in the jQuery object the value of the first successful one is returned. The *successful* argument controls whether or not the field element must be 'successful' (per <http://www.w3.org/TR/html4/interact/forms.html#successful-controls>). The default value of the *successful* argument is true. If this value is false then the value of the first field element in the jQuery object is returned. Note: If no valid value can be determined the return value will be undefined. Note: The *fieldValue* returned for a select-multiple element or for a checkbox input will always be an array if it is not undefined.

#### **Parameters**

**successful:** *Boolean*: true if value returned must be for a successful controls (default is true)

#### **Returns**

String or Array<String>

#### **Example**

Gets the current value of the myPasswordElement element

```
var data = $("#myPasswordElement").formValue();
```

#### **Example**

Get the value of the first successful control in the jQuery object.

```
var data = $("#myForm :input").formValue();
```

#### **Example**

Get the array of values for the first set of successful checkbox controls in the jQuery object.

```
var data = $("#myForm :checkbox").formValue();
```

#### **Example**

Get the value of the select control

```
var data = $("#mySingleSelect").formValue();
```

#### **Example**

Get the array of selected values for the select-multiple control

```
var data = $("#myMultiSelect").formValue();
```

## **fieldValue( el, successful )**

---

Returns the value of the field element. The *successful* argument controls whether or not the field element must be 'successful' (per <http://www.w3.org/TR/html4/interact/forms.html#successful-controls>). The default value of the *successful* argument is true. If the given element is not successful and the *successful* arg is not false then the returned value will be null. Note: The *fieldValue* returned for a select-multiple element will always be an array.

#### **Parameters**

**el:** *Element*: The DOM element for which the value will be returned

**successful:** *Boolean*: true if value returned must be for a successful controls (default is true)

**Returns**

String or Array<String>

**Example**

Gets the current value of the myPasswordElement element

```
var data = jQuery.fieldValue($("#myPasswordElement")[0]);
```

## clearForm( )

---

Clears the form data. Takes the following actions on the form's input fields: - input text fields will have their 'value' property set to the empty string - select elements will have their 'selectedIndex' property set to -1 - checkbox and radio inputs will have their 'checked' property set to false - inputs of type submit, button, reset, and hidden will *\*not\** be effected - button elements will *\*not\** be effected

**Returns**

jQuery

**Example**

Clears all forms on the page.

```
$('#form').clearForm();
```

## clearInputs( )

---

Clears the selected form elements. Takes the following actions on the matched elements: - input text fields will have their 'value' property set to the empty string - select elements will have their 'selectedIndex' property set to -1 - checkbox and radio inputs will have their 'checked' property set to false - inputs of type submit, button, reset, and hidden will *\*not\** be effected - button elements will *\*not\** be effected

**Returns**

jQuery

**Example**

Clears all inputs with class myInputs

```
$('.myInputs').clearInputs();
```

## resetForm( )

---

Resets the form data. Causes all form elements to be reset to their original value.

**Returns**

jQuery

**Example**

Resets all forms on the page.

```
$('#form').resetForm();
```

## Interface

---

### Draggable( hash )

---

Create a draggable element with a number of advanced options including callback, Google Maps type draggables, reversion, ghosting, and grid dragging.

**Parameters**

**hash:** *Hash*: A hash of parameters. All parameters are optional.

**Hash Options**

**handle:** *String*: The jQuery selector matching the handle that starts the draggable

**handle:** *DOMElement*: The DOM Element of the handle that starts the draggable

**revert:** *Boolean*: When true, on stop-drag the element returns to initial position

**ghosting:** *Boolean*: When true, a copy of the element is moved

**zIndex:** *Integer*: zIndex depth for the element while it is being dragged

**opacity:** *Float*: A number between 0 and 1 that indicates the opacity of the element while being dragged

**grid:** *Integer*: A number of pixels indicating the grid that the element should snap to

**grid:** *Array*: A number of x-pixels and y-pixels indicating the grid that the element should snap to

**fx:** *Integer*: Duration for the effect (like ghosting or revert) applied to the draggable

**containment:** *String*: Define the zone where the draggable can be moved. 'parent' moves it inside parent element, while 'document' prevents it from leaving the document and forcing additional scrolling

**containment:** *Array*: An 4-element array (topm left, width, height) indicating the containment of the element

**axis:** *String*: Set an axis: vertical (with 'vertically') or horizontal (with 'horizontally')

**onStart:** *Function*: Callback function triggered when the dragging starts

**onStop:** *Function*: Callback function triggered when the dragging stops

**onChange:** *Function*: Callback function triggered when the dragging stop \*and\* the element was moved at least one pixel

**onDrag:** *Function*: Callback function triggered while the element is dragged. Receives two parameters: x and y coordinates. You can return an object with new coordinates {x: x, y: y} so this way you can interact with the dragging process (for instance, build your containment)

**insideParent:** *Boolean*: Forces the element to remain inside its parent when being dragged (like Google Maps)

**snapDistance:** *Integer*: The element is not moved unless it is dragged more than snapDistance. You can prevent accidental dragging and keep regular clicking enabled (for links or form elements, for instance)

**cursorAt:** *Object*: The dragged element is moved to the cursor position with the offset specified. Accepts value for top, left, right and bottom offset. Basically, this forces the cursor to a particular position during the entire drag operation.

**autoSize:** *Boolean*: When true, the drag helper is resized to its content, instead of the dragged element's sizes

#### Returns

jQuery

## DraggableDestroy( )

---

Destroy an existing draggable on a collection of elements

#### Returns

jQuery

#### Example

```
$('#drag2').DraggableDestroy();
```

## Droppable( options )

---

With the Draggables plugin, Droppable allows you to create drop zones for draggable elements.

#### Parameters

**options:** *Hash*: A hash of options

#### Hash Options

**accept:** *String*: The class name for draggables to get accepted by the droppable (mandatory)

**activeclass:** *String*: When an acceptable draggable is moved, the droppable gets this class

**hoverclass:** *String*: When an acceptable draggable is inside the droppable, the droppable gets this class

**tolerance:** *String*: Choose from 'pointer', 'intersect', or 'fit'. The pointer options means that the pointer must be inside the droppable in order for the draggable to be dropped. The intersect option means that the draggable must intersect the droppable. The fit option means that the entire draggable must be inside the droppable.

**onDrop:** *Function:* When an acceptable draggable is dropped on a droppable, this callback is called. It passes the draggable DOMElement as a parameter.

**onHover:** *Function:* When an acceptable draggable is hovered over a droppable, this callback is called. It passes the draggable DOMElement as a parameter.

**onOut:** *Function:* When an acceptable draggable leaves a droppable, this callback is called. It passes the draggable DOMElement as a parameter.

**Example**

```
$('#dropzone1').Droppable( { accept : 'dropaccept', activeclass: 'dropzoneactive', hoverclass: 'dropzonehover', ondrop: function (drag) { alert(this); //the droppable alert(drag); //the draggable }, fit: true } )
```

---

## DroppableDestroy( )

Destroy an existing droppable on a collection of elements

**Returns**

jQuery

**Example**

```
$('#drag2').DroppableDestroy();
```

---

## \$.recallDroppables( )

Recalculate all Droppables

**Returns**

jQuery

**Example**

```
$.recallDroppable();
```

---

## Sortable( options )

Allows you to resort elements within a container by dragging and dropping. Requires the Draggables and Droppables plugins. The container and each item inside the container must have an ID. Sortables are especially useful for lists.

**Parameters**

**options:** *Hash:* A hash of options

**Hash Options**

**accept:** *String:* The class name for items inside the container (mandatory)

**activeclass:** *String:* The class for the container when one of its items has started to move

**hoverclass:** *String:* The class for the container when an acceptable item is inside it

**helperclass:** *String:* The helper is used to point to the place where the item will be moved. This is the class for the helper.

**opacity:** *Float:* Opacity (between 0 and 1) of the item while being dragged

**ghosting:** *Boolean:* When true, the sortable is ghosted when dragged

**tolerance:** *String:* Either 'pointer', 'intersect', or 'fit'. See Droppable for more details

**fit:** *Boolean:* When true, sortable must be inside the container in order to drop

**fx:** *Integer:* Duration for the effect applied to the sortable

**onchange:** *Function:* Callback that gets called when the sortable list changed. It takes an array of serialized elements

**floats:** *Boolean:* True if the sorted elements are floated

**containment:** *String:* Use 'parent' to constrain the drag to the container

**axis:** *String:* Use 'horizontally' or 'vertically' to constrain dragging to an axis

**handle:** *String*: The jQuery selector that indicates the draggable handle

**handle:** *DOMElement*: The node that indicates the draggable handle

**onHover:** *Function*: Callback that is called when an acceptable item is dragged over the container. Gets the hovering DOMElement as a parameter

**onOut:** *Function*: Callback that is called when an acceptable item leaves the container. Gets the leaving DOMElement as a parameter

**cursorAt:** *Object*: The mouse cursor will be moved to the offset on the dragged item indicated by the object, which takes "top", "bottom", "left", and "right" keys

**onStart:** *Function*: Callback function triggered when the dragging starts

**onStop:** *Function*: Callback function triggered when the dragging stops

#### Example

```
$('#ul').Sortable( { accept : 'sortableitem', activeclass : 'sortableactive', hoverclass : 'sortablehover', helperclass : 'sortheelper', opacity : 0.5, fit : false } )
```

#### See Also

Plugins/Interface/Draggable  
Plugins/Interface/Droppable

## SortableAddItem( elem )

---

A new item can be added to a sortable by adding it to the DOM and then adding it via SortableAddItem.

#### Parameters

**elem:** *DOMElement*: A DOM Element to add to the sortable list

#### Returns

jQuery

#### Example

```
$('#sortable1').append('<li id="newitem">new item</li>').SortableAddItem($('#new_item')[0])
```

## \$.SortSerialize( )

---

This function returns the hash and an object (can be used as arguments for \$.post) for every sortable in the page or specific sortables. The hash is based on the 'id' attributes of container and items.

#### Parameters

String sortable The id of the sortable to serialize

#### Returns

String

## Tabs

---

### tabs( initial, settings )

---

Create an accessible, unobtrusive tab interface based on a certain HTML structure. The underlying HTML has to look like this: `<div id="container"> <ul> <li><a href="#section-1">Section 1</a></li> <li><a href="#section-2">Section 2</a></li> <li><a href="#section-3">Section 3</a></li> </ul> <div id="section-1"> </div> <div id="section-2"> </div> <div id="section-3"> </div> </div>` Each anchor in the unordered list points directly to a section below represented by one of the divs (the URI in the anchor's href attribute refers to the fragment with the corresponding id). Because such HTML structure is fully functional on its own, e.g. without JavaScript, the tab interface is accessible and unobtrusive. A tab is also bookmarkable via hash in the URL. Use the History/Remote plugin (Tabs will auto-detect its presence) to fix the back (and forward) button.

#### Parameters

**initial:** *Number*: An integer specifying the position of the tab (no zero-based index) that gets first activated, e.g. on page load. If a hash in the URL of the page refers to one fragment (tab container) of a tab interface, this parameter will be ignored and instead the tab belonging to that fragment in that specific tab interface will be activated. Defaults to 1 if omitted.

**settings:** *Object*: An object literal containing key/value pairs to provide optional settings.

## Hash Options

**disabled**: *Array<Number>*: An array containing the position of the tabs (no zero-based index) that should be disabled on initialization. Default value: null.

**bookmarkable**: *Boolean*: Boolean flag indicating if support for bookmarking and history (via changing hash in the URL of the browser) is enabled. Default value: false, unless the History/Remote plugin is included. In that case the default value becomes true. @see \$.ajaxHistory.initialize

**fxFade**: *Boolean*: Boolean flag indicating whether fade in/out animations are used for tab switching. Can be combined with fxSlide. Will overrule fxShow/fxHide. Default value: false.

**fxSlide**: *Boolean*: Boolean flag indicating whether slide down/up animations are used for tab switching. Can be combined with fxFade. Will overrule fxShow/fxHide. Default value: false.

**fxSpeed**: *String|Number*: A string representing one of the three predefined speeds ("slow", "normal", or "fast") or the number of milliseconds (e.g. 1000) to run an animation. Default value: "normal".

**fxShow**: *Object*: An object literal of the form jQuery's animate function expects for making your own, custom animation to reveal a tab upon tab switch. Unlike fxFade or fxSlide this animation is independent from an optional hide animation. Default value: null. @see animate

**fxHide**: *Object*: An object literal of the form jQuery's animate function expects for making your own, custom animation to hide a tab upon tab switch. Unlike fxFade or fxSlide this animation is independent from an optional show animation. Default value: null. @see animate

**fxShowSpeed**: *String|Number*: A string representing one of the three predefined speeds ("slow", "normal", or "fast") or the number of milliseconds (e.g. 1000) to run the animation specified in fxShow. Default value: fxSpeed.

**fxHideSpeed**: *String|Number*: A string representing one of the three predefined speeds ("slow", "normal", or "fast") or the number of milliseconds (e.g. 1000) to run the animation specified in fxHide. Default value: fxSpeed.

**fxAutoHeight**: *Boolean*: Boolean flag that if set to true causes all tab heights to be constant (being the height of the tallest tab). Default value: false.

**onClick**: *Function*: A function to be invoked upon tab switch, immediately after a tab has been clicked, e.g. before the other's tab content gets hidden. The function gets passed three arguments: the first one is the clicked tab (e.g. an anchor element), the second one is the DOM element containing the content of the clicked tab (e.g. the div), the third argument is the one of the tab that gets hidden. Default value: null.

**onHide**: *Function*: A function to be invoked upon tab switch, immediately after one tab's content got hidden (with or without an animation) and right before the next tab is revealed. The function gets passed three arguments: the first one is the clicked tab (e.g. an anchor element), the second one is the DOM element containing the content of the clicked tab, (e.g. the div), the third argument is the one of the tab that gets hidden. Default value: null.

**onShow**: *Function*: A function to be invoked upon tab switch. This function is invoked after the new tab has been revealed, e.g. after the switch is completed. The function gets passed three arguments: the first one is the clicked tab (e.g. an anchor element), the second one is the DOM element containing the content of the clicked tab, (e.g. the div), the third argument is the one of the tab that gets hidden. Default value: null.

**selectedClass**: *String*: The CSS class attached to the li element representing the currently selected (active) tab. Default value: "tabs-selected".

**disabledClass**: *String*: The CSS class attached to the li element representing a disabled tab. Default value: "tabs-disabled".

**hideClass**: *String*: The CSS class used for hiding inactive tabs. A class is used instead of "display: none" in the style attribute to maintain control over visibility in other media types than screen, most notably print. Default value: "tabs-hide".

**tabStruct**: *String*: A CSS selector or basic XPath expression reflecting a nested HTML structure that is different from the default single div structure (one div with an id inside the overall container holds one tab's content). If for instance an additional div is required to wrap up the several tab containers such a structure is expressed by "div>div". Default value: "div".

## Returns

jQuery

## Example

Create a basic tab interface.

```
$('#container').tabs();
```

## Example

Create a basic tab interface with the second tab initially activated.

```
$('#container').tabs(2);
```

## Example

Create a tab interface with the third and fourth tab being disabled.

```
$('#container').tabs({disabled: [3, 4]});
```

## Example

Create a tab interface that uses slide down/up animations for showing/hiding tab content upon tab switching.

```
$('#container').tabs({fxSlide: true});
```

## triggerTab( position )

---

Activate a tab programmatically with the given position (no zero-based index), as if the tab itself were clicked.

### Parameters

**position:** *Number:* An integer specifying the position of the tab (no zero-based index) to be activated. If this parameter is omitted, the first tab will be activated.

### Returns

jQuery

### Example

Activate the second tab of the tab interface contained in <div id="container">.

```
$('#container').triggerTab(2);
```

### Example

Activate the first tab of the tab interface contained in <div id="container">.

```
$('#container').triggerTab(1);
```

### Example

Activate the first tab of the tab interface contained in <div id="container">.

```
$('#container').triggerTab();
```

## disableTab( position )

---

Disable a tab, so that clicking it has no effect.

### Parameters

**position:** *Number:* An integer specifying the position of the tab (no zero-based index) to be disabled. If this parameter is omitted, the first tab will be disabled.

### Returns

jQuery

### Example

Disable the second tab of the tab interface contained in <div id="container">.

```
$('#container').disableTab(2);
```

## enableTab( position )

---

Enable a tab that has been disabled.

### Parameters

**position:** *Number:* An integer specifying the position of the tab (no zero-based index) to be enabled. If this parameter is omitted, the first tab will be enabled.

### Returns

jQuery

### Example

Enable the second tab of the tab interface contained in <div id="container">.

```
$('#container').enableTab(2);
```

## Tooltip

---

## Tooltip( settings )

---

Display a customized tooltip instead of the default one for every selected element. The tooltip behaviour mimics the default one, but lets you style the tooltip and specify the delay before displaying it. In addition, it displays the href value, if it is available. To style the tooltip, use these selectors in your stylesheet: #tooltip - The tooltip container #tooltip h3 - The tooltip title #tooltip p.body - The tooltip body, shown when using showBody #tooltip p.url - The tooltip url, shown when using showURL

### Parameters

**settings:** *Object:* (optional) Customize your Tooltips

### Hash Options

**delay:** *Number:* The number of milliseconds before a tooltip is display, default is 250

**event:** *String:* The event on which the tooltip is displayed, default is "mouseover", "click" works fine, too

**track:** *Boolean:* If true, let the tooltip track the mousemovement, default is false

**showURL:** *Boolean:* If true, shows the href or src attribute within p.url, default is true

**showBody:** *String:* If specified, uses the String to split the title, displaying the first part in the h3 tag, all following in the p.body tag, separated with <br/>s, default is null

**extraClass:** *String:* If specified, adds the class to the tooltip helper, default is null

**fixPNG:** *Boolean:* If true, fixes transparent PNGs in IE, default is false

### Returns

jQuery

### Example

Shows tooltips for anchors, inputs and images, if they have a title

```
$( 'a, input, img' ).Tooltip();
```

### Example

Shows tooltips for labels with no delay, tracking mousemovement, displaying the tooltip when the label is clicked.

```
$( 'label' ).Tooltip( { delay: 0, track: true, event: "click" } );
```

### Example

This example starts with modifying the global settings, applying them to all following Tooltips; Afterwards, Tooltips for anchors with class pretty are created with an extra class for the Tooltip: "fancy" for anchors, "fancy-img" for images

```
// modify global settings $.extend( $.fn.Tooltip.defaults, { track: true, delay: 0, showURL: false, showBody: " - ", fixPNG: true } ); // setup fancy tooltips $( 'a.pretty' ).Tooltip( { extraClass: "fancy" } ); $( 'img.pretty' ).Tooltip( { extraClass: "fancy-img", } );
```

## Accordion

---

### Accordion( settings )

---

Make the selected elements Accordion widgets. ? Semantic requirements: If the structure of your container is flat with unique tags for header and content elements, eg. a definition list (dl > dt + dd), you don't have to specify any options at all. If your structure uses the same elements for header and content or uses some kind of nested structure, you have to specify the header elements, eg. via class, see the second example. Use activate(Number) to change the active content programmatically.

### Parameters

**settings:** *Object:* key/value pairs of optional settings.

### Hash Options

**active:** *String|Element|jQuery|Boolean:* Selector for the active element, default is the first child, set to false to display none at start

**header:** *String|Element|jQuery:* Selector for the header element, eg. div.title, a.head, default is the first child's tagname

**showSpeed:** *String|Number:* Speed for the slideIn, default is 'slow'

**hideSpeed:** *String|Number:* Speed for the slideOut, default is 'fast'

**selectedClass:** *String:* Class for active header elements, default is 'selected'

### Returns

jQuery

**Example**

Creates a Accordion from the given definition list

```
$('#list1').Accordion();
```

**Before**

```
<dl id="list1"><dt>Header 1<dd>Content 1</dd>[...]</dl>
```

**Example**

Creates a Accordion from the given div structure

```
$('#list2').Accordion({ header: 'div.title' });
```

**Before**

```
<div id="nav"><div><div class="title">Header 1</div><div>Content 1</div>[...]</div>
```

**Example**

Creates a Accordion from the given navigation list

```
$('#nav').Accordion({ header: 'a.head' });
```

**Before**

```
<ul id="nav"> <li> <a class="head">Header 1 </li> <li><a href="#">Link 1</a></li> <li><a href="#">Link 2</a></li> </ul> [...] </ul>
```

**Example**

Updates the #status element with the text of the selected header every time the accordion changes

```
$('#accordion').Accordion().change(function(event, newHeader, oldHeader, newContent, oldContent) {  
    $('#status').html(newHeader.text()); });
```

## activate( index )

---

Activate a content part of the Accordion programmatically with the position zero-based index. If the index is not specified, it defaults to zero, if it is an invalid index, eg. a string, nothing happens. Requires jQuery core revision >= 557.

**Parameters**

**index:** *Number:* An Integer specifying the zero-based index of the content to be activated. Defaults to 0.

**Returns**

jQuery

**Example**

Activate the second content of the Accordion contained in <div id="accordion">.

```
$('#accordion').activate(1);
```

**Example**

Activate the first content of the Accordion contained in <ul id="nav">.

```
$('#nav').activate();
```