Contents

Preface

The creation of this book has been for me, Mike McGrath, an exciting opportunity to build upon the scripting examples demonstrated in the companion book "JavaScript in easy steps". Employing the jQuery JavaScript library lets you easily create dynamic interactive functionality. Example code listed in this book describes how to produce jQuery functionality in easy steps – and the screenshots illustrate the actual results. I sincerely hope you enjoy discovering the exciting possibilities of jQuery and have as much fun with it as I did in writing this book.

In order to clarify the code listed in the steps given in each example I have adopted certain colorization conventions. Components of the JavaScript language are generally colored blue, HTML code and literal values are black, and comments are green. Additionally, in an attempt to readily identify HTML components selected by a jQuery object, those components are colored red – both in the listed HTML code and in the listed JavaScript code, like this:

```
<p> <button id = "btn" > Send AJAX Request </button> </p>

$( document ).ready( function() {
  $( "#btn" ).on( "click" , function() {
  // AJAX request to be added here.
  } ) ;
) } ;
```

In order to identify each source code file described in the steps a colored icon and a file name appears in the margin alongside the steps:

page.html

plugin.js

parser.php

product.txt

For convenience I have placed source code files from the examples featured in this book into a single ZIP archive. You can obtain the complete archive by following these three easy steps:

1. Open a web browser and navigate to **www.ineasysteps.com** then select the "Free Resources" menu and choose the "Downloads" item

2. Next find "jQuery in easy steps" in the list then click on the hyperlink entitled "All Code Examples" to download the archive

3. Now extract the contents to any convenient location, such as your Desktop, and copy all contents of the **htdocs** sub-folder into your web server's documents directory

1 Getting started

Welcome to the exciting world of jQuery scripting for dynamic web pages. This chapter introduces the jQuery library and demonstrates how to select HTML elements.

Introducing jQuery

jQuery is a lightweight multi-browser JavaScript library that is designed to make it easier to incorporate JavaScript features on websites. The library provides methods that can be called with just a single line of code to achieve results that would otherwise require many lines of JavaScript – so you can "write less, do more".

jQuery is free open-source software that was first released back in 2006 by its original author John Resig, but is now maintained by the jQuery Team of developers. The library is currently used by over 65% of the 10,000 most-visited websites and is the most popular JavaScript library in use today.

jQuery's syntax is designed to make it easy to select components of a web page for the purposes of:

- Manipulation of HTML/DOM elements
- Modification of CSS style rules
- Attachment of event handlers to respond to user actions
- Creation of dynamic effects and animations
- Development of interactive AJAX applications

jQuery also allows developers to create their own plugins on top of the standard JavaScript library to suit specific requirements. These may provide anything from a simple, small utility to a complex widget or an advanced animation. This modular approach to the jQuery library enables the creation of powerful dynamic web pages and web applications.

jQuery is supported by the Adobe Dreamweaver web development application and by the Microsoft Visual Studio Express For Web development application for use within the ASP.NET AJAX and ASP.NET MVC frameworks. Both applications have an "Intellisense" feature that recognizes jQuery syntax and provides coding hints. This feature is very useful but it is not at all essential – all the examples in this book are easily created in a plain text editor, such as Windows' Notepad application.

Don't forget

JavaScript is a client-side scripting language whereas jQuery is a library written in the JavaScript language.

Enabling jQuery

The entire jQuery library is contained in a single JavaScript file that may be freely downloaded from **jquery.com/download** Compressed and uncompressed copies of the jQuery library file are available for download. There are no functional differences between these files but the uncompressed copy is formatted with spacing and comments, to make it easily human-readable, whereas the compressed copy is minimized by the removal of spacing and comments – to reduce the file size and so improve performance. Unless you wish to examine the source code of the jQuery library itself you should download the compressed copy and place it in the directory containing your HTML web page documents.

In order to enable the jQuery library to be used on your web pages the jQuery library file must be incorporated into the web page by including an HTML **<script>** tag within the head section of the HTML document to indicate the jQuery library's location.

The minimized jQuery library is a file named something like **jquery-1.11.0.min.js**, describing the current version number, and can be enabled in your web page by including these HTML tags in the head section of your web page:

```
<script src="jquery-1.11.0.min.js"> </script>
```

There are also 2.x.x versions of the jQuery library file that are further minimized by removal of support for older web browsers. As an alternative to hosting a copy of the jQuery library alongside your web pages you can indicate the jQuery library's location online at the Google CDN (Content Delivery Network) like this:

```
<script
   src="//ajax.googleapis.com/ajax/libs/jquery/1.11.0/jquery.min.js">
</script>
```

or at the Microsoft CDN (Content Delivery Network) like this:

```
<script
   src="//ajax.aspnetcdn.com/ajax/jQuery/jquery-1.11.0.min.js">
</script>
```

As jQuery is so popular many users may already have downloaded the library file from Google or Microsoft into their browser's cache when visiting another site. This means it can quickly be loaded from cache when they visit your site.

jQuery resolves multi-browser issues automatically – it performs identically in all major browsers, including Internet Explorer 6.

Change the version number in the tag to match that of the jQuery library file you download or indicate online.

Recognizing readiness

Having enabled jQuery to be used on your web page, as described on the previous page, you can add a second **<script>** element to the head section to contain statements calling jQuery functions. Each jQuery statement typically contains a "selector", to select one or more HTML elements of the document, and an "action" to be performed on that selection. The statement syntax looks like this:

$(***selector*** **).*action*( ) ;**

The **$( )** function queries the HTML document to find the elements specified by the selector then returns a jQuery object that is a collection of those matched elements. Dot notation is then used to append the action to be performed on the elements.

A comprehensive collection of elements specified by a selector cannot be accurately retrieved until the HTML document has loaded and its Document Object Model (DOM) has been built. At this point a "document ready" event gets fired in the browser to signal that the document is ready for manipulation by jQuery.

Readiness of the document can be recognized by specifying the document's own **document** object as the selector and calling the **ready()** action function when the "document ready" event fires.

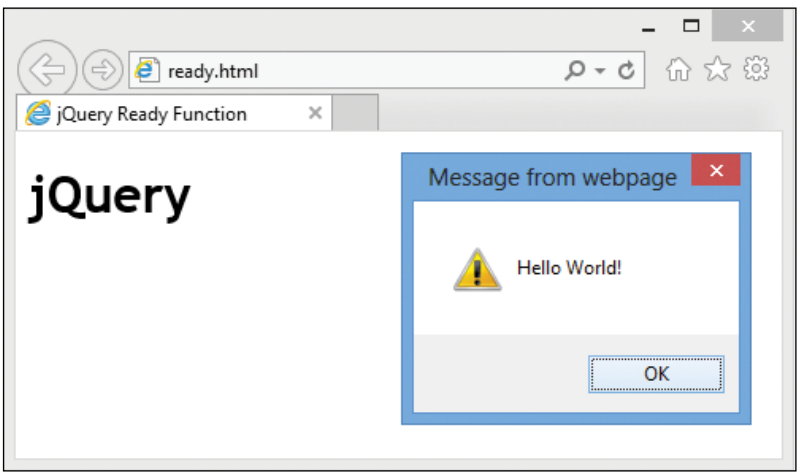
It is good practice to specify an anonymous JavaScript function within the parentheses of the **ready()** action function whose statements can be executed safe in the knowledge that all elements are available. The complete code block looks like this:

```
<script>

$( document ).ready( function() {

// Statements to be executed when the document is ready.

} ) ;

</script>
```

Every example in this book places statements within the body of this anonymous JavaScript function to ensure the document is completely loaded before attempting to manipulate its content. The HTML5 document listed opposite makes the jQuery library available then simply produces an alert dialog indicating readiness.

It is not necessary to include a **type** attribute in the **<script>** tag for HTML5 documents.

```
< !DOCTYPE HTML >
< html lang = "en" >
< head >
< meta charset = "UTF-8" >
< title >jQuery Ready Function< /title >

< script src = "jquery-1.11.0.min.js" >< /script >

< script >

$( document ).ready(  function() {

  alert( "Hello World!" ) ;

} ) ;

< /script >
< /head >
< body >
< h1 >jQuery< /h1 >
<!-- Elements to be added here. -->
< /body >
< /html >
```



As all statements are always wrapped inside the call to the **$(document).ready()** function jQuery also allows this shorter alternative syntax to be used if you prefer it:

```
$( function() {

// Statements to be executed when the document is ready.

} ) ;
```

ready.html

Hot tip

The examples in this book use this HTML5 document format merely changing the highlighted page title, statements in the anonymous function body, and elements in the document body.

Beware

The call to recognize the document ready event is more easily understood using the longer syntax so the shorter alternative is not used for the examples in this book.

Avoiding conflict

The "$" character that is used by jQuery syntax is also used by other JavaScript libraries such as MooTools, Backbone and JavaScriptMVC. It is therefore possible for conflicts to arise between calls to jQuery functions and those of another library.

Fortunately the **$()** function syntax is merely a shortcut to the actual library function **jQuery()** which may be used directly instead. Additionally, the jQuery library provides a **jQuery.noConflict()** method that removes the shortcut ability of the "$" character so conflicts between libraries can be avoided:

noconflict.html

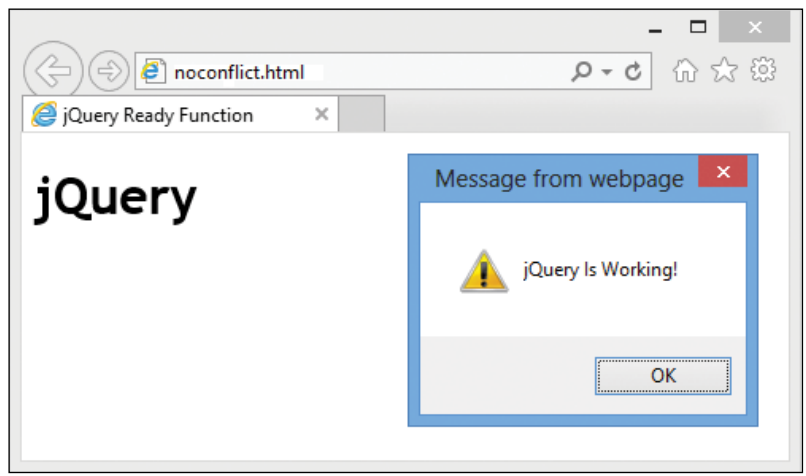
1. In the head section of an HTML document begin a script block with a call to remove the **$()** shortcut ability

```
jQuery.noConflict() ;
```

2. Next, add a direct call to recognize the "document ready" event by producing an alert dialog message

```
jQuery( document ).ready( function() {

  alert( "jQuery Is Working!" ) ;
} ) ;
```

3. Save the document alongside the jQuery library file then open it in a web browser to see the direct call execute



Hot tip

Change the dialog call to **alert(jQuery.jquery) ;** to see the jQuery library version number.

Don't forget

The **$()** and **jQuery()** calls are normally interchangeable so you can use either. The shorter **$()** call is used throughout the examples in this book for brevity.

Selecting by tag name

Element Selector

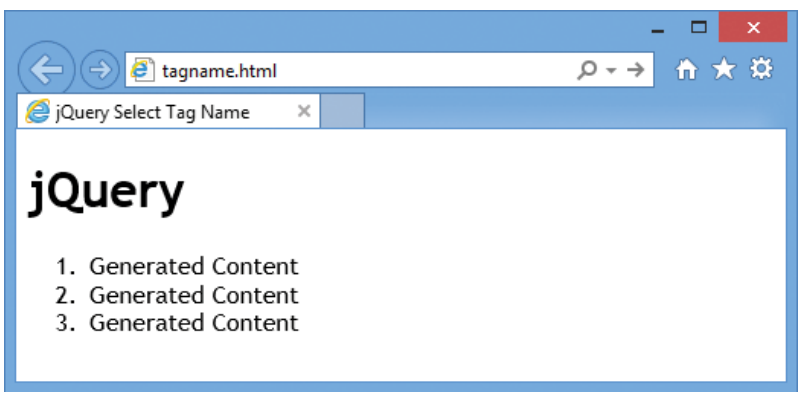
A jQuery selector specifies which elements of the HTML document are to be returned in the jQuery object for manipulation. An Element Selector simply specifies a tag name within quote marks between the selector function's parentheses. For example, the selector call **$("p")** queries the HTML document and returns all **<p>** paragraph elements.

Each selector call can append an action function, using dot notation, to specify an action to perform on the collection of returned elements. For example, the jQuery **text()** method can be appended to specify a text string within quote marks between its parentheses. The specified string will then be written into each matched element, replacing any existing content. This could be used to replace the existing text content of each item in a list:

1. Add an ordered list to the body of an HTML document
```
<ol><li>Item<li>Item<li>Item</ol>
```

2. In the head section of the HTML document add a script block that recognizes the "document ready" event by selecting all list item elements and replacing text content
```
$( document ).ready( function() {

  $( "li" ).text( "Generated Content" ) ;
} ) ;
```

3. Save the document alongside the jQuery library file then open it in a browser to see the generated list item content



tagname.html

Only specify the tag name to the selector – angled markup brackets are not required.

Selecting by ID and class

ID Selector

A jQuery selector can specify a particular element by its identity. An ID Selector specifies the unique value of an element that has been assigned to its **id** attribute. This must be prefixed by a **#** hash character and enclosed within quote marks between the selector function's parentheses. For example, a selector **$("#two")** queries the HTML document and returns the element that has the value "two" assigned to its **id** attribute. An identity selector call can append an action function, using dot notation, to specify an action to perform on the returned element:

identity.html

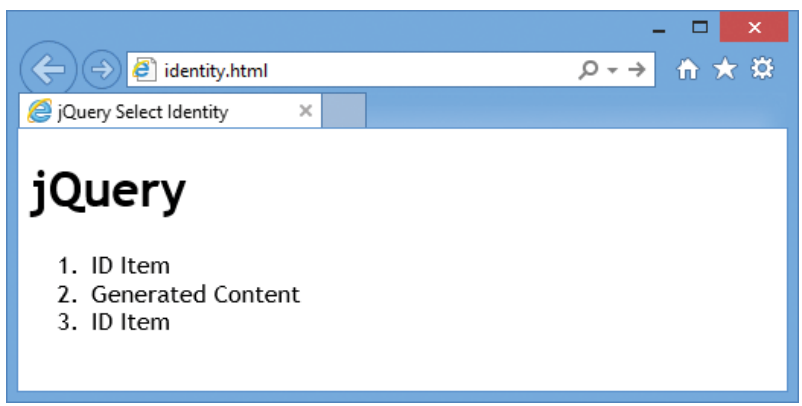
1. Add an ordered list to the body of an HTML document

```
<ol>
<li id = "one" >ID Item
<li id = "two" >ID Item
<li id = "three" >ID Item
</ol>
```

2. In the head section of the HTML document add a script block that recognizes the "document ready" event by selecting one list item element and replacing text content

```
$( document ).ready( function() {

  $( "#two" ).text( "Generated Content" ) ;
} ) ;
```

3. Save the document alongside the jQuery library file then open it in a browser to see the generated list item content



Hot tip

Matching by identity internally uses the **getElementById()** DOM method and is a fast and efficient technique.

Class Selector

A jQuery Class Selector can also return a collection of elements of a particular class by specifying the value assigned to their **class** attribute. This should be prefixed by the element name followed by a . period character, and the whole expression enclosed within quote marks between the selector function's parentheses. For example, the selector **$("li.rest")** queries the HTML document and returns the list item elements that have the value "rest" assigned to their **class** attribute. A class selector call can append an action function, using dot notation, to specify an action to perform on the returned elements:

1. Add an ordered list to the body of an HTML document

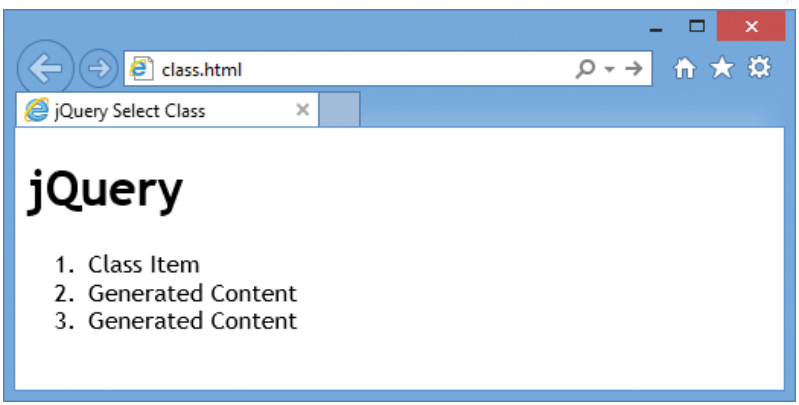
```
<ol>
<li class = "start" >Class Item
<li class = "rest" >Class Item
<li class = "rest" >Class Item
</ol>
```

class.html

2. In the head section of the HTML document add a script block that recognizes the "document ready" event by selecting list item elements and replacing text content

```
$( document ).ready( function() {

  $( "li.rest" ).text( "Generated Content" ) ;
} ) ;
```

3. Save the document alongside the jQuery library file then open it in a browser to see the generated list item content



Internally, element selectors use the **getElementsByTagName()** DOM method and class selectors use the **getElementsByClassName()** DOM method.

Selecting attributes and order

Attribute selector

A jQuery selector can specify elements by attribute name or by attribute name and value. An "Has Attribute" selector selects all elements having an attribute name specified in **[ ]** brackets within double quote marks between the selector function's parentheses. For example, **$("[lang]")** returns all elements that have a "lang" attribute. An "Equals" selector also specifies an attribute value – enclosed within single quote marks to differentiate it from the outer double quotes. For example, **$("[lang='fr']")** returns all elements that have the value "fr" assigned to their "lang" attribute. Attribute selector calls can append a method, using dot notation, to specify an action to perform on the returned elements:

attribute.html

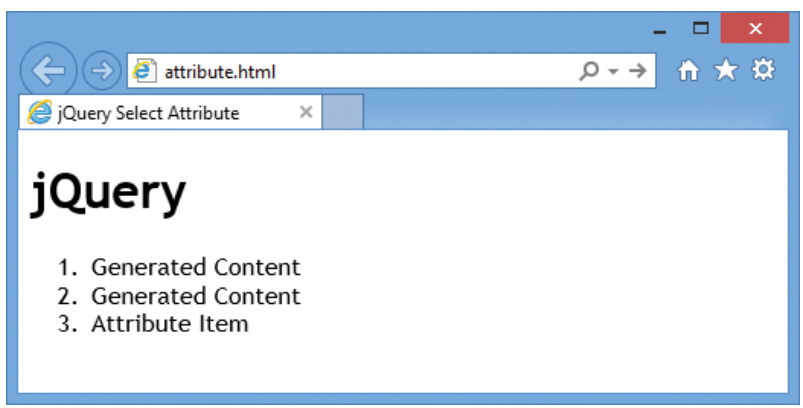
1. Add an ordered list to the body of an HTML document

```
<ol>
<li lang = "fr" >Attribute Item
<li lang = "fr" >Attribute Item
<li lang = "en" >Attribute Item
</ol>
```

2. In the head section of the HTML document add a script block that recognizes the "document ready" event by selecting list item elements and replacing text content

```
$( document ).ready( function() {

  $( "[lang='fr']" ).text( "Generated Content" ) ;
} ) ;
```

3. Save the document alongside the jQuery library file then open it in a browser to see the generated list item content



Hot tip

The selector can prefix the attribute by a tag name to select only elements of a certain type having the specified attribute. For example, **$("p[lang]")** returns only paragraph elements that have a **lang** attribute.

…cont’d

Filter selector

A jQuery selector can also return elements according to their zero-based index position in a selection. The element name can be suffixed by **:first** or **:last** “filters” to select a single element. Alternatively, the filters **:odd** or **:even** can be used to select a subset. An individual element can also be selected by specifying its index number within the brackets of an **:eq( )** filter. Subsets of the collection can be selected above and below a specified index position by specifying that index number within the brackets of **:gt( )** or **:lt( )** filters. For example, a selector **$(“li:lt(2)”)** queries the HTML document and returns the first and second elements. An order selector call can append an action method, using dot notation, to specify an action to perform on returned elements:

Hot tip

The **:odd** filter can be used to stripe table rows.

1. Add an unordered list to the body of an HTML document

```
<ul>
<li>Even Item 0<li>Odd Item 1<li>Even Item 2
</ul>
```

order.html

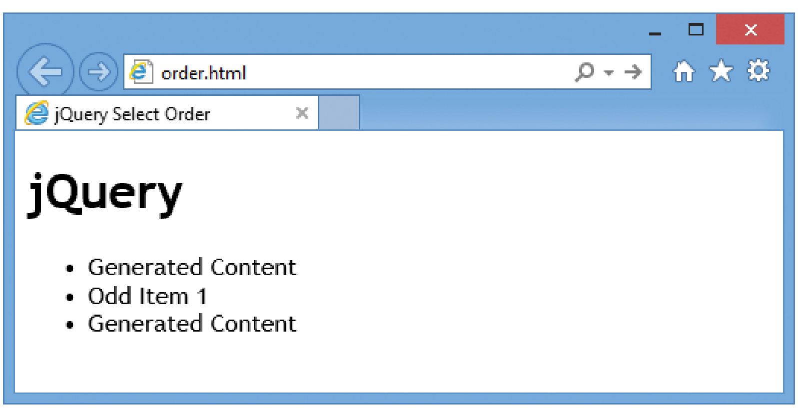
2. In the head section of the HTML document add a script block that recognizes the “document ready” event by selecting list item elements and replacing text content

```
$( document ).ready( function() {

  $( “li:even” ).text( “Generated Content” ) ;
} ) ;
```

3. Save the document alongside the jQuery library file then open it in a browser to see the generated list item content

Don’t forget

Index numbering begins at zero and is considered to be an even number.



Selecting family relatives

Descendant selector

A jQuery selector can return elements according to their relationship to another element using a "Descendant" selector to specify the outer parent element name followed by a space and the name of an inner descendant element. For example, the selector **$("p span")** queries the HTML document and returns all span elements within each paragraph of the document. Descendant selector calls can append an action method, using dot notation, to specify an action to perform on returned elements:

descendant.html

1. Add an ordered list to the body of an HTML document

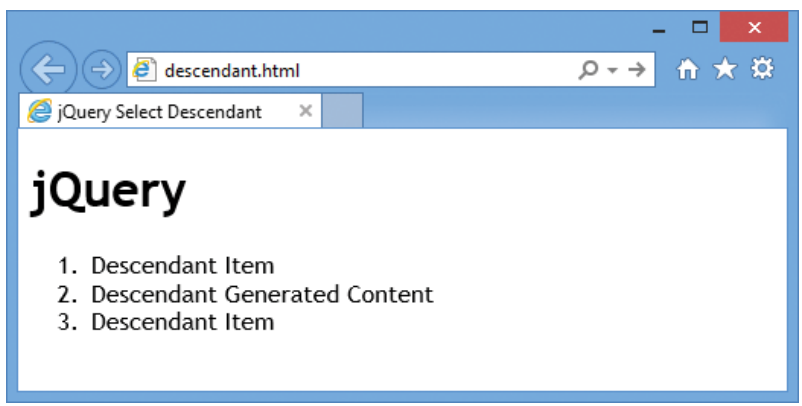
```
<ol>
<li>Descendant Item
<li>Descendant <span>Item</span>
<li>Descendant Item
</ol>
```

2. In the head section of the HTML document add a script block that recognizes the "document ready" event by selecting a span element and replacing its text content

```
$( document ).ready( function() {

  $( "ol span" ).text( "Generated Content" ) ;
} ) ;
```

3. Save the document alongside the jQuery library file then open it in a browser to see the generated span content

Descendant selectors select immediate child elements and other hierarchy levels too, such as grandchild elements.



Child selector

All child elements of a specified parent element can be selected using a "Child" selector stating the parent name followed by a ">" character and the child name. For example, a selector **$("ol > li")** selects all list items in each ordered list of the HTML document.

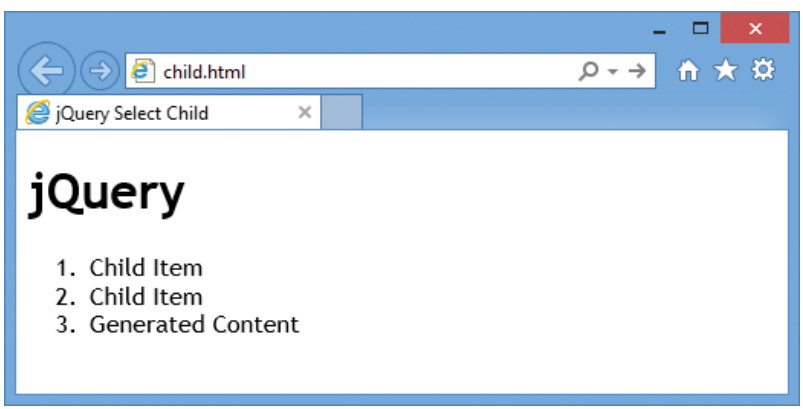
Child filter selector

A jQuery selector can also specify elements by their relationship to other elements using a Child Filter specifying the name of a child element followed by a **:first-child** or **:last-child** filter to select a single child of each parent element. For example, a selector **$("li:first-child")** selects the first child list item element within each list of the HTML document. Child selectors and selectors with Child Filters can append a method, using dot notation, to specify an action to perform on the returned elements:

1. Add an ordered list to the body of an HTML document
```
<ol><li>Child Item<li>Child Item<li>Child Item</ol>
```

2. In the head section of the HTML document add a script block that recognizes the "document ready" event by selecting a list item element and replacing its text content
```
$( document ).ready( function() {

  $( "li:last-child" ).text( "Generated Content" ) ;
} ) ;
```

3. Save the document alongside the jQuery library file then open it in a browser to see the generated list item content



Beware

Unlike the Descendant Selector a Child Selector selects only immediate child elements, not grandchild elements.

child.html

Hot tip

You can also use **:first** and **:last** filters to select only the first and last single elements within an HTML document. For example, **$("p:first")** selects only the very first paragraph.

Selecting multiple elements

Multiple selector

A jQuery selector can be combined with another to create a specific selection expression. For example, an identity selector that targets a particular list in an HTML document can be combined with an "Order" selector that targets a particular item in that list. Additionally, multiple selection expressions can appear in a single jQuery selector as a comma-separated list. Each multiple selector can append an action method, using dot notation, to specify an action to perform on all the returned elements:

multiple.html

1. Add two ordered lists to the body of an HTML document, each with a unique identity

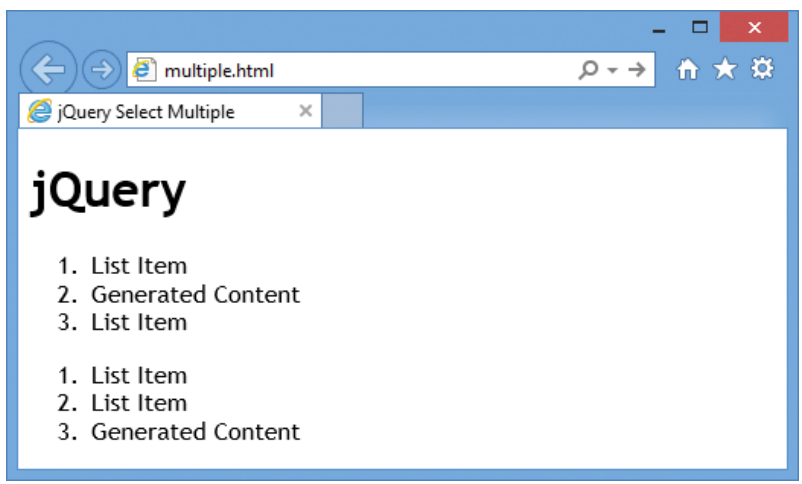
```
<ol id = "list1" >
<li>List Item<li>List Item<li>List Item</ol>
<ol id = "list2" >
<li>List Item<li>List Item<li>List Item</ol>
```

2. In the head section of the HTML document add a script block that recognizes the "document ready" event by selecting an item in each list and replacing text content

```
$( document ).ready( function() {

$( "#list1 li:eq(1) , #list2 li:eq(2)" ).text( "Generated Content" ) ;
} ) ;
```

3. Save the document alongside the jQuery library file then open it in a browser to see the generated list item content

Index numbering begins at zero so the second item in a list is at index position one.



All selector

In jQuery the * asterisk character is a wildcard that can be used in a **$("*")** universal "All" selector to select all elements in an HTML document. Perhaps more usefully, it can also be used with the jQuery **find()** method to select all descendants of a specified element. For example, a selector **$("body").find("*")** queries the HTML document and returns all elements within the document body. The returned collection can be assigned to a variable, creating an array, and a loop can be used to reveal each tag name:

1. Add a paragraph containing a span after the heading in the body of an HTML document

```
<h1>jQuery</h1>
<p>Output: <span id = "out" ></span></p>
```

universal.html

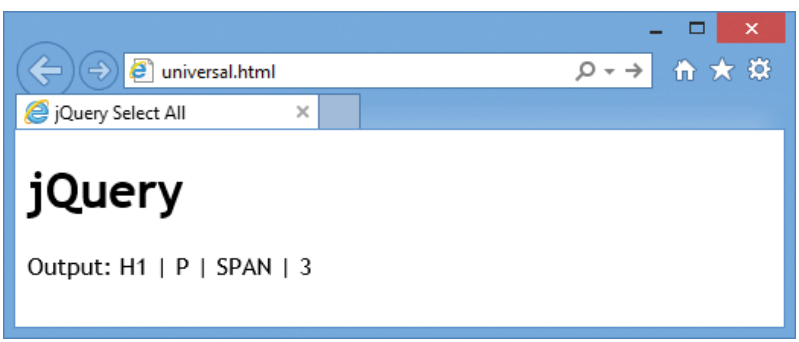
2. In the head section of the HTML document add a script block that recognizes the "document ready" event by first initializing counter, string, and array variables

```
$( document ).ready( function() {

  var i = 0 , str = "" , elements = $( "body" ).find( "*" ) ;
} ) ;
```

3. Next, in the script block, add a loop to build a string of retrieved tag names then display that string in the span

```
for( ; i < elements.length ; i++ )
{ str += ( elements[ i ].tagName + " | " ) } ;
$( "#out" ).text( str + elements.length ) ;
```

4. Save the document alongside the jQuery library file then open it in a browser to see the generated span content



The jQuery documentation warns that the * universal selector is "extremely slow, except when used by itself".

Filtering visibility

Visibility filter selector

A jQuery selector can filter out elements from a returned collection according to their visibility on the page by specifying **:visible** or **:hidden** filters to the jQuery **filter()** method. Only elements that have styles of **display:none**, form elements of **type="hidden"**, elements whose width and height are set to zero, or who are descendants of any of these, are regarded as hidden. For example, a selector **$("body").find("*").filter(":hidden")** returns all truly hidden elements within the document body:

hidden.html

1. Add a hidden division with bold text and a paragraph containing a span in the body of an HTML document

```
<div style="display:none"><b>Hidden Text</b></div>
<p>Output: <span id = "out" ></span></p>
```

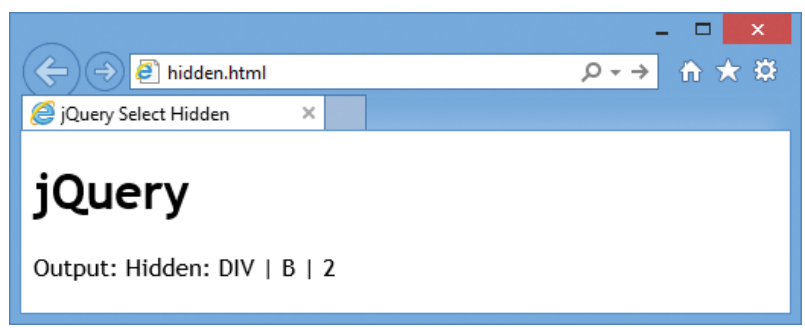
2. In the head section of the HTML document add a script block that recognizes the "document ready" event by first initializing counter, string, and array variables

```
$( document ).ready( function() {

  var i = 0 , str = "Hidden: " ;
  var hid = $( "body" ).find( "*" ).filter( ":hidden" ) ;
} ) ;
```

3. Next, in the script block, add a loop to build a string of hidden tag names then display that string in the span

```
for( ; i < hid.length ; i++ )
{ str += ( hid[ i ].tagName + " | " ) } ;
$( "#out" ).text( str + hid.length ) ;
```

4. Save the document alongside the jQuery library file then open it in a browser to see the generated span content



Elements that have styles of **visibility:hidden** or **opacity:0** are regarded as visible because they still reserve page space.

Optimizing selections

Examples on the previous pages demonstrate various ways to select a collection of elements with jQuery but some consideration should be given to selector efficiency:

- **Select by identity where possible** – HTML **id** attributes are unique in each page so even older browsers find the element quickly. Ideally use **$("#theIdentity")** format for each selector.
- **Select with both element and class** – when selecting by **class** older browsers examine every element on the page unless the selector gives an element name to limit the search.
  Use **$("p.theClass")** format rather than **$(".theClass")**
- **Select in the simplest way** – avoid unnecessary complexity.
  A complex selector **$("body article.topic p#theIdentity em")** can simply use **$("#theIdentity em")** instead as the **id** is unique anyway so including its ancestors is unnecessary.
- **Select from right to left** – jQuery selects from right to left so it loads all right-most matches into an array before discarding those that do not match those to the left in the selector.
  A selector **$("#theIdentity em")** loads all document **em** elements before discarding those not matching **#theIdentity**.
  For optimum efficiency use **$("#theIdentity").find("em")**
- **Select only once** – avoid repetition of selections like these
  $("p").text("Generated Content") ;
  $("p").css("color" , "red") ;
  "Chaining" can apply multiple methods in one statement.
  Use **$("p").text("Generated Content").css("color" , "red") ;**
  Assign the jQuery object to a variable if you need to reference it more than once like the statement in the example opposite
  var hid = $("body").find("*").filter(":hidden") ;
  Then, use **hid.length** and **hid[i].tagName** as often as required.

Unlike a **class** selector, which should include the element name, an identity selector should not include the element name – just the **id** value.

Chaining is the technique of appending multiple jQuery method calls to a single selector using dot notation and should be used wherever possible.

Summary

- jQuery is a lightweight multi-browser JavaScript library that makes it easier to incorporate JavaScript features on websites
- jQuery can be enabled by hosting the minimized jQuery library file on your website or alternatively by indicating the library file's location online at the Google or Microsoft CDN
- A jQuery statement typically contains a selector to return a collection of elements and an action to be performed on them
- The **$(document).ready( )** function gets called when the DOM has loaded and so can usefully contain an anonymous JavaScript function enclosing all jQuery statements
- The **$( )** function syntax is a shortcut for the **jQuery( )** function that can be used instead, and if using **jQuery.noConflict()**
- jQuery's most basic selectors specify elements by tag name **$("p")**, by identity **$("#theId")**, and by class **$("p.theClass")**
- jQuery Attribute selectors select all elements containing a set attribute **$("[lang]")** or those of set value **$("[lang='fr']")**
- jQuery Order selectors select elements according to their zero-based index position in a selected collection **$("li:eq(1)")**
- jQuery Child selectors select only all immediate children of a specified parent element, but not grandchildren **$("ol > li")**
- jQuery Descendant selectors select all hierarchy levels below a specified parent element, including grandchildren **$("p span")**
- Multiple selection expressions can appear in a single jQuery selector as a comma-separated list
- The * universal selector can be used with the **find()** method to select all descendants of an element **$("body").find("*")**
- Elements can be selected according to visibility with the **filter()** method **$("body").find("*").filter(":hidden")**