# JQUERY PROGRAMMING COOKBOOK

Hot Recipes for jQuery Development

jQuery

write less, do more.

FABIO CIMO

WEB CODE GEEKS
WEB DEVELOPERS RESOURCE CENTER

# jQuery Programming Cookbook

# Contents

# Preface

jQuery is a cross-platform JavaScript library designed to simplify the client-side scripting of HTML. jQuery is the most popular JavaScript library in use today, with installation on 65% of the top 10 million highest-trafficked sites on the Web.

jQuery's syntax is designed to make it easier to navigate a document, select DOM elements, create animations, handle events, and develop Ajax applications. jQuery also provides capabilities for developers to create plug-ins on top of the JavaScript library. This enables developers to create abstractions for low-level interaction and animation, advanced effects and high-level, theme-able widgets. The modular approach to the jQuery library allows the creation of powerful dynamic web pages and web applications. (Source: https://en.wikipedia.org/wiki/JQuery)

In this ebook, we provide a compilation of jQuery based examples that will help you kick-start your own web projects. We cover a wide range of topics, from UI Widgets, to Drag and Drop functionality and CSS manipulation. With our straightforward tutorials, you will be able to get your own projects up and running in minimum time.

# About the Author

Fabio is a passionate student in web tehnologies including front-end (HTML/CSS) and web design. He likes exploring as much as possible about the world wide web and how it can be more productive for us all. Currently he studies Computer Engineering, at the same time he works as a freelancer on both web programming and graphic design.

# Chapter 1

# Add/Remove Class Example

In this example, we'll go through jQuery `.addClass()` and `.removeClass()` methods. jQuery provides a seamless, easy and efficent way to add or remove classes to specific DOM elements and trigger these events on the various listening events it provides like `click`, `mouseover`, `mouseleave` ect.

These methods are very useful to dynamically change content based on your cases, and gives your website a rather interactive and engaging user experience. From changing styles and colors to animations, it provides powerful ways to get you going.

You'll be able to access these methods just by including the jQuery library in your HTML document.

## 1.1 Basic Document Setup

To begin, create a new HTML document and add the following basic syntax inside:

```
<!DOCTYPE html>
<html>
<head>
        <title>jQuery Add/Remove Class Example</title>
</head>
<body>

<!-- STYLE SECTION  -->
<style type="text/css">

</style>


<!-- HTML SECTION  -->


<!-- JAVASCRIPT SECTION  -->
<script src="jquery-1.11.3.min.js"></script>

<script type="text/javascript">

</script>

</body>
</html>
```

Also, let's create some elements with classes and define properties for these classes in CSS. For now, there will be only one element and two classes:

```
<!-- HTML SECTION  -->
I am going to get classes added or removed!<div>
```

For now, we'd have this view:

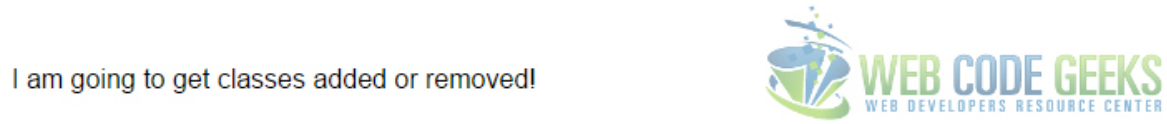


Figure 1.1: Basic Element View

Alongside with this element, let's add two classes, one for styling and another for shaping:

```
<!-- STYLE SECTION  -->
<style type="text/css">
    .style {
        font-family: "Clear Sans";
        font-size: 1.5em;
        color: #AE0001;
        font-style: oblique;
        padding: 1em;
    }
    .shape {
        width: 25em;
        height: 3em;
        border: 0.1em solid #2c3e50;
        margin: 5em;
        line-height: 3em;
        text-align: center;
    }
    .decoration {
        text-decoration: underline;
        font-size: 1.2em;
        padding: 1em;
        width: 15em;
    }
    .color {
        color: #e74c3c;
    }
    .background {
        background-color: #ecf0f1;
        font-size: 1.2em;
        padding: 1em;
        width: 15em;
    }
</style>
```

Next, we'll continue adding and removing classes that we have.

## 1.2 .addClass() and .removeClass() Methods

### 1.2.1 Add Class

The `addClass()` method adds the specified class(es) to each element in the set of matched elements:

1. `.addClass("className")`, in which `className` is a string containing one or more space-separated classes to be added to the class attribute of each matched element.
2. `.addClass(function)` in which `className` A function returning one or more space-separated class names to be added to the existing class name(s). Receives the index position of the element in the set and the existing class name(s) as arguments. Within the function, this refers to the current element in the set.

Now using jQuery, we can add a class to our `.element`:

```
<!-- JAVASCRIPT SECTION  -->
<script type="text/javascript">
    $('.element').addClass('style');
</script>
```

Easy as that, just selected the DOM element and added a class.

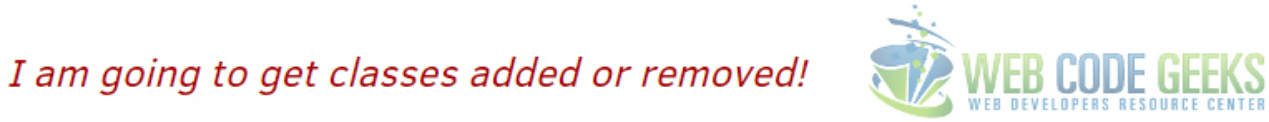


Figure 1.2: Element View after Adding the *style* Class

We added the *.style* class, the same goes for *.shape* class and we'd have this view:

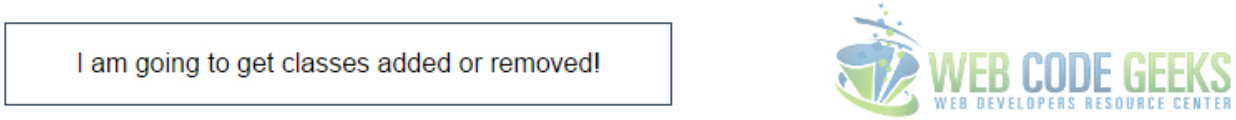


Figure 1.3: Added the *.shape* Class

But what is interesting, is that you can have more than one class added, just by leaving a blank space between them:

```
<!-- JAVASCRIPT SECTION  -->
<script type="text/javascript">
    $('.element').addClass('style shape');
</script>
```


Figure 1.4: Multiple Classes Added

### 1.2.2 Remove Class

Similarily, .removeClass() removes a (or some) class/es from an element. To show this, add a new HTML element:

```
<!-- HTML SECTION  -->
I need to get rid of some classes
```

In the JS section, let's remove two of these classes:

```
<!-- JAVASCRIPT SECTION  -->
<script type="text/javascript">
    $('.element').addClass('style shape');
</script>
```

The view we'd get is this:


Figure 1.5: Remove Class Method

## 1.3 Adding or Removing Classes on Event Listeners

You can trigger classes only when a specific event listener happens to become true. Let's add some elements:

```
<!-- HTML SECTION  -->
<h2 class="decoration">This is a click event add class.</h2>
```

```
<!-- JAVASCRIPT SECTION  -->
<script type="text/javascript">
    $('article').hide();
    $('h2').click(function(e){
        e.preventDefault();
        $(this).addClass('shape color');
        $(this).removeClass('decoration');
    });
</script>
```

The result would be two classes added and one removed after click like so:

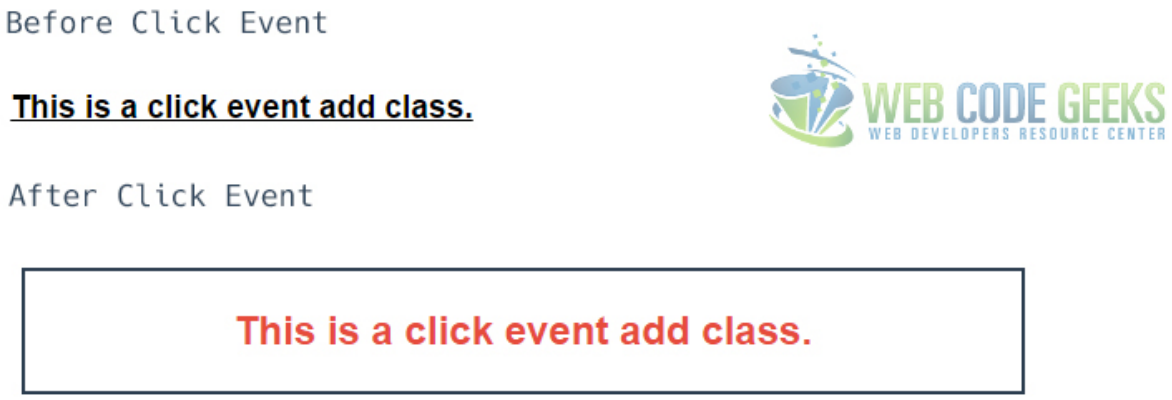


Figure 1.6: Trigger Add or Remove Class on Event Listener!

### 1.3.1 Toggle Classes on Event Listeners

You can use `.toggleClass()` to toggle between the two states of an element, with and without classes like this:

```
<!-- JAVASCRIPT SECTION  -->
<script type="text/javascript">
    $('article').hide();
    $('h2').click(function(e){
        e.preventDefault();
        $(this).toggleClass('shape color');
    });
</script>
```

Check out the functionality here.

## 1.4 Conclusion

To conclude, the jQuery methods for adding or removing classes are just right whenever it feels useful. You can actually add or remove content (by adding or removing classes), change the view on mouse or keyboard event ect. It is a process where you can try a lot and see how it best fits to what you want to achieve. I use it a lot when adding classes from a third party CSS stylesheet like an animation one, to animate elements `mouseover` or `mouseleave`. It is up to you.

## 1.5 Download

**Download** You can download the full source code of this example here: **jQuery Add/Remove Class**

# Chapter 2

# UI Datepicker

In this example, we'll have a look at the `datepicker` widget of jQuery. The jQuery UI Datepicker is a highly configurable plugin that adds datepicker functionality to your pages. You can customize the date format and language, restrict the selectable date ranges and add in buttons and other navigation options easily.

By default, the datepicker calendar opens in a small overlay when the associated text field gains focus. For an inline calendar, simply attach the datepicker to a div or span.

There are quite some other javascript frameworks out there that offer the `datepicker` widget better designed, but that is up to you to decide.

## 2.1 Basic Setup & Application

The following sections will help you begin with the very basics.

### 2.1.1 Document Setup

To begin, create a new HTML document and add the following basic syntax to it:

```
<!DOCTYPE html>
<html>
<head>
        <title>jQuery Datepicker Example</title>
</head>
<body>

<!-- LINKS SECTION  -->
<link href="http://ajax.googleapis.com/ajax/libs/jqueryui/1.11.4/themes/smoothness/jquery- ↩
    ui.css" rel="stylesheet" type="text/css"/>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js"></script>
<script src="http://ajax.googleapis.com/ajax/libs/jqueryui/1.11.4/jquery-ui.min.js"></ ↩
    script>
<link href="style.css" rel="stylesheet" type="text/css"/>

<!-- HTML SECTION  -->


<!-- JAVASCRIPT SECTION  -->
<script src="jquery-1.11.3.min.js"></script>

<script type="text/javascript">
```

```
</script>

</body>
</html>
```

Don't forget to download or link jQuery library, otherwise the code won't work. Other links of the jQuery UI are provided, so you don't have to.

### 2.1.2 Default Functionality

Let us begin with this very simple and basic datepicker. The datepicker is tied to a standard form input field. Focus on the input (click, or use the tab key) to open an interactive calendar in a small overlay. Choose a date, click elsewhere on the page (blur the input), or hit the Esc key to close. If a date is chosen, feedback is shown as the input's value.

So, create a new `p` element in HTML and add some text like Date:. Inside the `p` add an `input` element and give it a class of `.datepicker`:

```
<!-- HTML SECTION  -->
Date: <input type="text" class="datepicker">
```

Now, to show a basic datepicker, in jQuery, create a new function where you select the .datepicker input field and add the `.datepicker()` method.

```
<!-- JAVASCRIPT SECTION  -->
<script type="text/javascript">
  $(function() {
    $( ".datepicker" ).datepicker();
  });
</script>
```

Check out the functionality here.

## 2.2 Options and Examples

The following section will expand on the various customizations the widget can have.

### 2.2.1 Animations

You can use different animations when opening or closing the datepicker. Choose an animation from the dropdown, then click on the input to see its effect. You can use one of the three standard animations or any of the UI Effects.

```
<!-- HTML SECTION  -->
Date: <input type="text" class="datepicker" size="30">

Animations:<br>
  <select class="anim">
    <option value="show">Show (default)</option>
    <option value="slideDown">Slide down</option>
    <option value="fadeIn">Fade in</option>
    <option value="blind">Blind (UI Effect)</option>
    <option value="bounce">Bounce (UI Effect)</option>
    <option value="clip">Clip (UI Effect)</option>
    <option value="drop">Drop (UI Effect)</option>
    <option value="fold">Fold (UI Effect)</option>
    <option value="slide">Slide (UI Effect)</option>
    <option value="">None</option>
  </select>
```

```
<!-- JAVASCRIPT SECTION  -->
<script type="text/javascript">
  $(function() {
    $( ".datepicker" ).datepicker();
    $( ".anim" ).change(function() {
      $( ".datepicker" ).datepicker( "option", "showAnim", $(this).val() );
    });
  });
</script>
```

Look at the results in this video: datepicker-2

### 2.2.2 Dates in Other Months

You might have noticed that the calendar does not show dates that are not of the current month. You can change that using the `showOtherMonths` and `selectOtherMonths` options. Just add these two lines inside your `.datepicker()` method.

```
<!-- JAVASCRIPT SECTION  -->
<script type="text/javascript">
  $(function() {
    $( ".datepicker" ).datepicker({
        showOtherMonths: true,
        selectOtherMonths: true
    });
  });
</script>
```

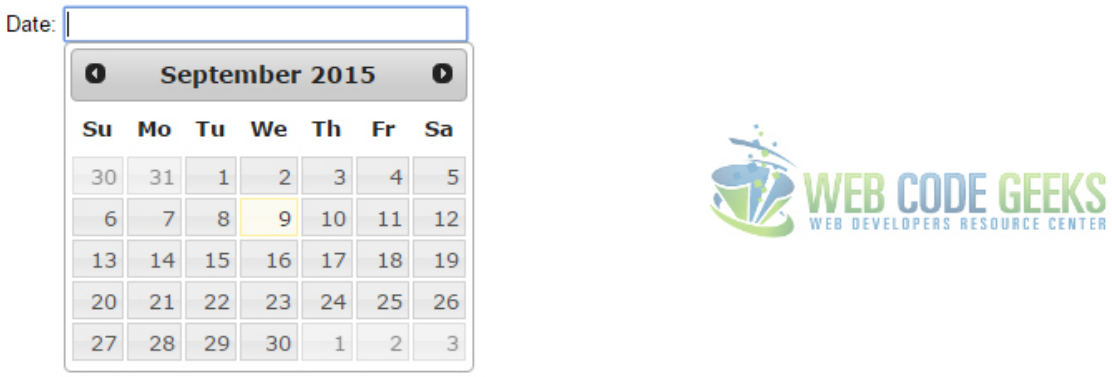


Figure 2.1: Dates in other months!

### 2.2.3 Display Button Bar

Display a button for selecting Today's date and a Done button for closing the calendar with the boolean `showButtonPanel` option. Each button is enabled by default when the bar is displayed, but can be turned off with additional options. Button text is customizable.

```
<!-- JAVASCRIPT SECTION  -->
<script type="text/javascript">
  $(function() {
```

```
    $( ".datepicker" ).datepicker({
        showButtonPanel: true
    });
  });
</script>
```

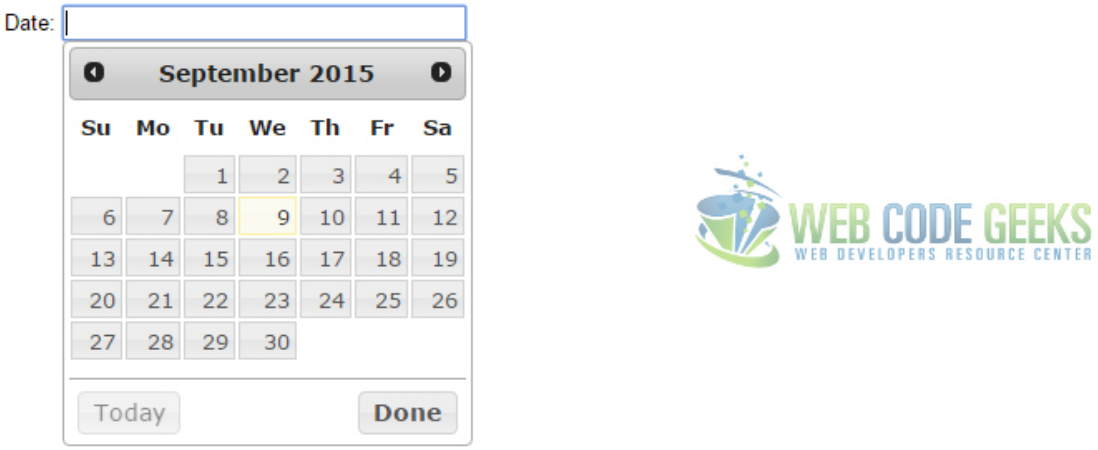


Figure 2.2: Display a button bar!

### 2.2.4 Display Month and Year Menus

Show month and year dropdowns in place of the static month/year header to facilitate navigation through large timeframes. Add the boolean `changeMonth` and `changeYear` options.

```
<!-- JAVASCRIPT SECTION  -->
<script type="text/javascript">
    $(function() {
    $( ".datepicker" ).datepicker({
      changeMonth: true,
      changeYear: true
    });
  });
</script>
```


Figure 2.3: Display Month & Year Menus!

### 2.2.5 Display Multiple Months

Set the `numberOfMonths` option to an integer of 2 or more to show multiple months in a single datepicker.

```
<!-- JAVASCRIPT SECTION  -->
<script type="text/javascript">
    $(function() {
    $( ".datepicker" ).datepicker({
      numberOfMonths: 3
    });
  });
</script>
```


Figure 2.4: Select Multiple Months!

### 2.2.6 Select a Date Range

Select the date range to search for.

```
<!-- HTML SECTION  -->
<label for="from">From</label>
<input type="text" id="from" name="from">
<label for="to">to</label>
<input type="text" id="to" name="to">
```

```
<!-- JAVASCRIPT SECTION  -->
<script type="text/javascript">
  $(function() {
    $( "#from" ).datepicker({
      defaultDate: "+1w",
      changeMonth: true,
      numberOfMonths: 3,
      onClose: function( selectedDate ) {
        $( "#to" ).datepicker( "option", "minDate", selectedDate );
      }
    });
    $( "#to" ).datepicker({
      defaultDate: "+1w",
      changeMonth: true,
      numberOfMonths: 3,
      onClose: function( selectedDate ) {
        $( "#from" ).datepicker( "option", "maxDate", selectedDate );
      }
    });
  });
</script>
```

Look at the results in this video: Video-9-9-2015-12-10-44-PM

### 2.2.7 Icon Trigger

Click the icon next to the input field to show the datepicker. Set the datepicker to open on focus (default behavior), on icon click, or both.

```
<!-- JAVASCRIPT SECTION  -->
<script type="text/javascript">
  $(function() {
    $( ".datepicker" ).datepicker({
      showOn: "button",
      buttonImage: "calendar.png",
      buttonImageOnly: true,
      buttonText: "Select date"
    });
  });
</script>
```

Check out the functionality here.

### 2.2.8 Format Date

Display date feedback in a variety of ways. Choose a date format from the dropdown, then click on the input and select a date to see it in that format.

```
<!-- HTML SECTION  -->
Format options:<br>
  <select class="format">
    <option value="mm/dd/yy">Default - mm/dd/yy</option>
```

```
    <option value="yy-mm-dd">ISO 8601 - yy-mm-dd</option>
    <option value="d M, y">Short - d M, y</option>
    <option value="d MM, y">Medium - d MM, y</option>
    <option value="DD, d MM, yy">Full - DD, d MM, yy</option>
    <option value="'day' d 'of' MM 'in the year' yy">With text - 'day' d 'of' MM 'in the ↩
        year' yy</option>
  </select>
```

```
<!-- JAVASCRIPT SECTION  -->
<script type="text/javascript">
    $(function() {
    $( ".datepicker" ).datepicker();
    $( ".format" ).change(function() {
      $( ".datepicker" ).datepicker( "option", "dateFormat", $(this).val() );
    });
  });
</script>
```

Look at the results in this video: datepicker-9

## 2.3 Conclusion

To conclude, the `datepicker` widget of jQuery UI is a complete solution for developers whenever it comes to getting a date information or a period of time from the user. The datepicker is very used in airlines, hotels and other reservation websties. It is easy to implement and use in jQuery and you can use jQuery UI themes to have a different design.

## 2.4 Download

**Download** You can download the full source code of this example here: **jQuery Datepicker**

# Chapter 3

# jQuery and AJAX

The aim of this example is to give you a full understanding of AJAX, which stands for Asynchronous Javascript and XML. Ajax is not a programming language or a tool, but a concept.

Ajax is a client-side script that communicates to and from a server/database without the need for a postback or a complete page refresh.

The best definition for Ajax would be "the method of exchanging data with a server, and updating parts of a web page - without reloading the entire page."

Ajax itself is mostly a generic term for various JavaScript techniques used to connect to a web server dynamically without necessarily loading multiple pages. In a more narrowly-defined sense, it refers to the use of XmlHttpRequest objects to interact with a web server dynamically via JS.

## 3.1 An Introduction to AJAX!

### 3.1.1 How we came here?

Traditionally webpages required reloading to update their content. For web-based email this meant that users had to manually reload their inbox to check and see if they had new mail. This had huge drawbacks: it was slow and it required user input. When the user reloaded their inbox, the server had to reconstruct the entire web page and resend all of the HTML, CSS, JavaScript, as well as the user's email.

This was hugely inefficient. Ideally, the server should only have to send the user's new messages, not the entire page. By 2003, all the major browsers solved this issue by adopting the XMLHttpRequest (XHR) object, allowing browsers to communicate with the server without requiring a page reload.

The XMLHttpRequest object is part of a technology called Ajax (Asynchronous JavaScript and XML). Using Ajax, data could then be passed between the browser and the server, using the XMLHttpRequest API, without having to reload the web page. With the widespread adoption of the XMLHttpRequest object it quickly became possible to build web applications like Google Maps, and Gmail that used XMLHttpRequest to get new map tiles, or new email without having to reload the entire page.

Ajax requests are triggered by JavaScript code; your code sends a request to a URL, and when it receives a response, a callback function can be triggered to handle the response. Because the request is asynchronous, the rest of your code continues to execute while the request is being processed, so it's imperative that a callback be used to handle the response.

Unfortunately, different browsers implement the Ajax API differently. Typically this meant that developers would have to account for all the different browsers to ensure that Ajax would work universally. Fortunately, jQuery provides Ajax support that abstracts away painful browser differences. It offers both a full-featured `$.ajax()` method, and simple convenience methods such as `$.get()`, `$.getScript()`, `$.getJSON()`, `$.post()`, and `$().load()`.

### 3.1.2 Getting to Know AJAX Better

Like DHTML and LAMP, AJAX is not a technology in itself, but a group of technologies. AJAX uses a combination of:

- HTML and CSS for marking up and styling information.
- The DOM accessed with JavaScript to dynamically display and interact with the information presented.
- A method for exchanging data asynchronously between browser and server, thereby avoiding page reloads. The XMLHttpRequest (XHR) object is usually used, but sometimes an IFrame object or a dynamically added tag is used instead.
- A format for the data sent to the browser. Common formats include XML, pre-formatted HTML, plain text, and JavaScript Object Notation (JSON). This data could be created dynamically by some form of server-side scripting.

A picture being worth a thousand words, below a diagram that illustrates the communication between the client and the remote server, as well as the differences between the classic and the AJAX-powered applications:

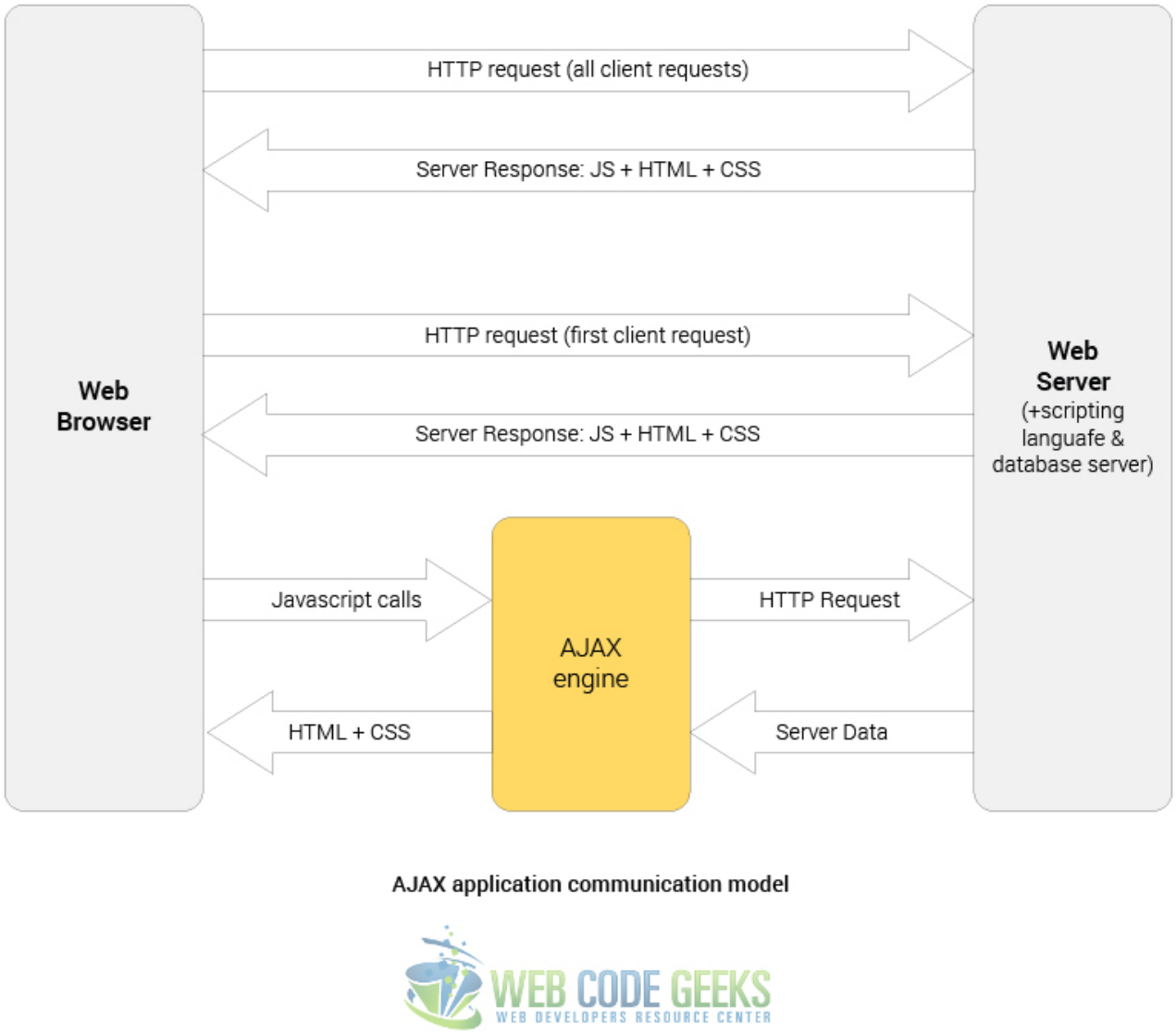


Figure 3.1: AJAX Application Communication Model

For the orange part, you can do everything by hand (with the `XMLHttpRequest` object) or you can use famous JavaScript libraries like jQuery, Prototype, YUI, etc to "AJAXify" the client-side of your application. Such libraries aim to hide the complexity of JavaScript development (e.g. the cross-browser compatibility), but might be overkill for a simple feature.

On the server-side, some frameworks can help too (e.g. DWR or RAJAX if you are using Java), but all you need to do is basically to expose a service that returns only the required informations to partially update the page (initially as XML/XHTML - the X in AJAX - but JSON is often preferred nowadays).

### 3.1.3 AJAX Benefits

There are 4 main benefits of using Ajax in web applications:

1. **Callbacks**: Ajax is used to perform a callback, making a quick round trip to and from the server to retrieve and/or save data without posting the entire page back to the server. By not performing a full postback and sending all form data to the server, network utilization is minimized and quicker operations occur. In sites and locations with restricted bandwidth, this can greatly improve network performance. Most of the time, the data being sent to and from the server is minimal. By using callbacks, the server is not required to process all form elements. By sending only the necessary data, there is limited processing on the server. There is no need to process all form elements, process the ViewState, send images back to the client, or send a full page back to the client.
2. **Making Asynchronous Calls**: Ajax allows you to make asynchronous calls to a web server. This allows the client browser to avoid waiting for all data to arrive before allowing the user to act once more.
3. **User-Friendly**: Because a page postback is being eliminated, Ajax enabled applications will always be more responsive, faster and more user-friendly.
4. **Increased Speed**: The main purpose of Ajax is to improve the speed, performance and usability of a web application. A great example of Ajax is the movie rating feature on Netflix. The user rates a movie and their personal rating for that movie will be saved to their database without waiting for the page to refresh or reload. These movie ratings are being saved to their database without posting the entire page back to the server.

## 3.2 Implementation

To begin, set up your document, and get to learn how to organize your files for this to work.

### 3.2.1 Basic Document Setup

To begin, create a new HTML document and add the basic syntax inside it like so:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
        <title>jQuery AJAX Example</title>
</head>
<body>

<!-- HTML SECTION  -->

<!-- JAVASCRIPT SECTION  -->
<script src="jquery-1.11.3.min.js"></script>

<script type="text/javascript">

</script>

</body>
</html>
```

We will create another HTML file, where we'll add some random content that we'd like to retrieve later with AJAX:

```
<!DOCTYPE html>
<html>
<head>
        <title>jQuery AJAX Example</title>
</head>
<body>

<!-- HTML SECTION  -->
<div class="content">
    <h2>I just showed up here!</h2>
    I am a paragraph, just retrieved with AJAX!
    <img src="image.png">


</body>
</html>
```

Here, we don't need the javascript section at all. This is how this page looks like for now:

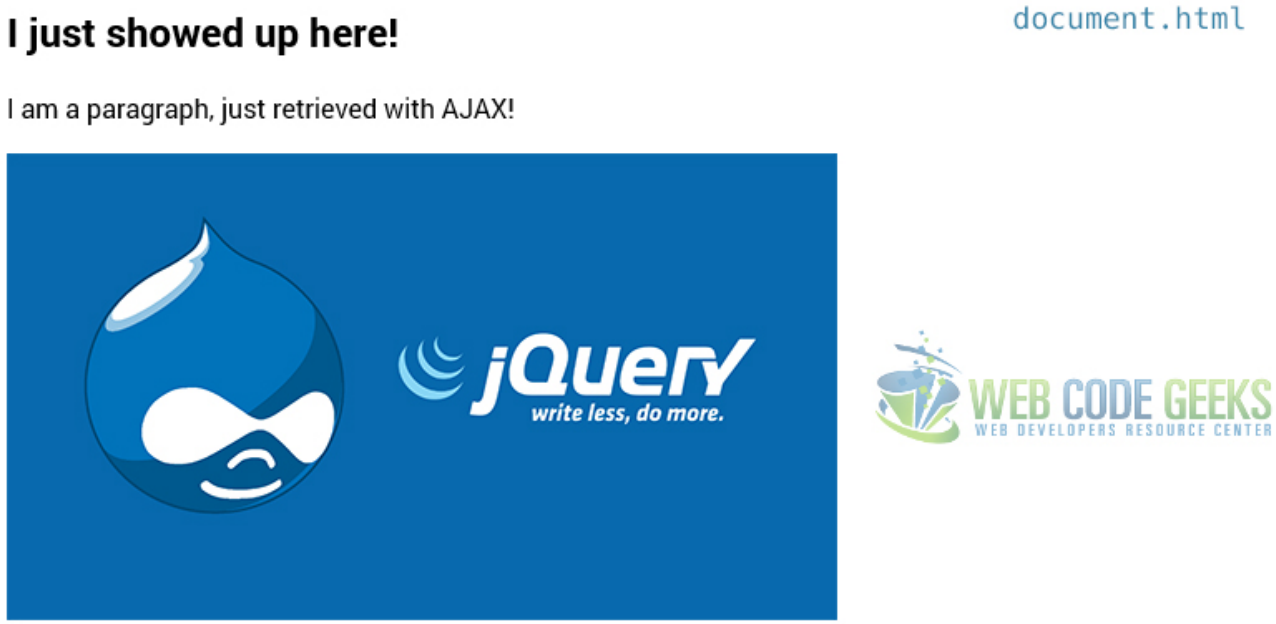


Figure 3.2: An extra HTML where content is going to be retrieved.

### 3.2.2 Ajax Declaration

The basix syntax in which you can start using AJAX is:

`jQuery.ajax('url', {settings})` or simply `$.ajax('url', {settings})` where:

1. **url** is of a type `string` and contains the url to where the request is sent.
2. **settings** is of a type `PlainObject` and can be considered set of key/value pairs that configure the Ajax request.

This is how a simple AJAX request would look like:

```
$.ajax('file.html', {
            success: function(){
            /* do sth if the file.html is reached successfully */
            },
            type: 'GET'
});
```

As you can see, in this simple example, we only configured two settings, because settings are optional, and you can set as many as you like referring to the existing ones. The `success` function is going to do sth if the request is accepted and data can be retrieved, while `type` is telling AJAX that this is a request and not a submission on the server. We'll have a more extensive look at settings later in this article.

You can achieve the same using a shorthand method which is `$.get(url, success);` like this:

```
$.get('document.html', function(response){
        $('.content').html(response).slideDown();
});
```

### 3.2.3 A Real-World AJAX Example!

We've already set up our basic HTML docs, now let's add some content on the first one:

```
<!-- HTML SECTION  -->

    <button>Click Me</button>   <!-- content will be shown on this button click -->
    <div class="content"> <!-- content will be shown in this div -->
</div>
```

Now, on the JavaScript section, we're going to listen for a click on the button we just created and then show any element we want that is located on the remote `document.html` file: (look at the image below the source code for explanation)

```
<!-- JAVASCRIPT SECTION  -->
<script src="jquery-1.11.3.min.js"></script>

<script type="text/javascript">
    $('button').on('click', function(){
        $.ajax('document.html', {
            success: function(response){
             $('.content').html(response);
            },
            type: 'GET',
    });
    });
</script>
```


Figure 3.3: AJAX - Retrieving Data!

This would mean that AJAX will call all elements found in `document.html` and show them just at the moment of clicking the button like so.

Check out the functionality here.

Additionally, you can show only part of a HTML document (that is, you can filter to show only section of a website). Let's take the example above, where we have three different tag types that contain their own data, we ahve `h2`, a paragraph `p` and an image `img`. If you want to show only one of them, or two you can modify your ajax call to search the DOM for the response and there find the wanted tag (or class if we referred to classes):

```
<!-- JAVASCRIPT SECTION  -->
<script src="jquery-1.11.3.min.js"></script>

<script type="text/javascript">
    $('button').on('click', function(){
        $.ajax('document.html', {
            success: function(response){
             $('.content').html($(response).find('img, h2').fadeIn());
            },
            type: 'GET',
    });
    });
</script>
```

Only the line where we put data under the `.content` div is changed to get both the image and h2 element on button click and fade them in, but not the paragraph. Now see it in action.

Check out the functionality here.

## 3.3 AJAX Settings

Below, there is basic information about the most important of the ajax settings that you can use.

### 3.3.1 accepts

- default: `depends on DataType`
- type: `PlainObject`
- function: The content type sent in the request header that tells the server what kind of response it will accept in return.

### 3.3.2 async

- default: `true`
- type: `Boolean`
- function: By default, all requests are sent asynchronously (i.e. this is set to `true` by default). If you need synchronous requests, set this option to `false`. Cross-domain requests and `dataType:"jsonp"` requests do not support synchronous operation. Note that synchronous requests may temporarily lock the browser, disabling any actions while the request is active.

### 3.3.3 beforeSend

- default: does not apply
- type: `Function`
- function: A pre-request callback function that can be used to modify the jqXHR (in jQuery 1.4.x, XMLHTTPRequest) object before it is sent. Use this to set custom headers, etc. The jqXHR and settings objects are passed as arguments. This is an Ajax Event. Returning `false` in the `beforeSend` function will cancel the request.

### 3.3.4 cache

- default: `true, false for dataType script and jsonp`
- type: `Boolean`
- function: If set to `false`, it will force requested pages not to be cached by the browser. Note: Setting `cache` to false will only work correctly with HEAD and GET requests. It works by appending timestamp to the GET parameters. The parameter is not needed for other types of requests, except in IE8 when a POST is made to a URL that has already been requested by a GET.

### 3.3.5 complete

- default: does not apply
- type: `Function`
- function: A function to be called when the request finishes (after `success` and `error` callbacks are executed). The function gets passed two arguments: The jqXHR (in jQuery 1.4.x, XMLHTTPRequest) object and a string categorizing the status of the request (`"success"`, `"notmodified"`, `"nocontent"`, `"error"`, `"timeout"`, `"abort"`, or `"parsererror"`).

### 3.3.6 contents

- default: does not apply
- type: `PlainObject`
- function: An object of string/regular-expression pairs that determine how jQuery will parse the response, given its content type.

### 3.3.7 contentType

- default: `'application/x-www-form-urlencoded;charset=UTF-8'`
- type: `Boolean`, `String`
- function: When sending data to the server, use this content type. Default is "application/x-www-form-urlencoded; charset=UTF-8", which is fine for most cases. If you explicitly pass in a content-type to $.ajax(), then it is always sent to the server (even if no data is sent).

### 3.3.8 context

- default: does not apply
- type: `PlainObject`
- function: This object will be the context of all Ajax-related callbacks. By default, the context is an object that represents the Ajax settings used in the call (`$.ajaxSettings` merged with the settings passed to `$.ajax`). For example, specifying a DOM element as the context will make that the context for the `complete` callback of a request, like so:

```
$.ajax({
  url: "test.html",
  context: document.body
}).done(function() {
  $( this ).addClass( "done" );
});
```

### 3.3.9 data

- default: does not apply
- type: `PlainObject`, `String`, `Array`
- function: Data to be sent to the server. It is converted to a query string, if not already a string. It's appended to the url for GET-requests. See `processData` option to prevent this automatic processing. Object must be Key/Value pairs. If value is an Array, jQuery serializes multiple values with same key based on the value of the traditional setting (described below).

### 3.3.10 data

- default: does not apply
- type: `Function`
- function: A function to be used to handle the raw response data of XMLHttpRequest. This is a pre-filtering function to sanitize the response. You should return the sanitized data. The function accepts two arguments: The raw data returned from the server and the *dataType* parameter.

### 3.3.11 dataType

- default: (`xml`, `json`, `script`, or `html`)
- type: `String`
- function: The type of data that you're expecting back from the server. If none is specified, jQuery will try to infer it based on the MIME type of the response (an XML MIME type will yield XML, in 1.4 JSON will yield a JavaScript object, in 1.4 script will execute the script, and anything else will be returned as a string). The available types (and the result passed as the first argument to your success callback) are:
- `"xml"`: Returns a XML document that can be processed via jQuery.
- `"html"`: Returns HTML as plain text; included script tags are evaluated when inserted in the DOM.
- `"script"`: Evaluates the response as JavaScript and returns it as plain text. Disables caching by appending a query string parameter, `_=[TIMESTAMP]`, to the URL unless the `cache` option is set to `true`. Note: This will turn POSTs into GETs for remote-domain requests.
- `"json"`: Evaluates the response as JSON and returns a JavaScript object. Cross-domain

- `"jsonp"`: Loads in a JSON block using JSONP. Adds an extra `"?callback=?"` to the end of your URL to specify the callback. Disables caching by appending a query string parameter, `"_=[TIMESTAMP]"`, to the URL unless the `cache` option is set to `true`.
- `"text"`: A plain text string.

### 3.3.12 error

- default: does not apply
- type: `Function`
- function: A function to be called if the request fails. The function receives three arguments: The jqXHR (in jQuery 1.4.x, XMLHttpRequest) object, a string describing the type of error that occurred and an optional exception object, if one occurred. Possible values for the second argument (besides `null`) are `"timeout"`, `"error"`, `"abort"`, and `"parsererror"`. When an HTTP error occurs, `errorThrown` receives the textual portion of the HTTP status, such as "Not Found" or "Internal Server Error."

### 3.3.13 global

- default: `true`
- type: `Boolean`
- function: Whether to trigger global Ajax event handlers for this request. The default is `true`. Set to `false` to prevent the global handlers like `ajaxStart` or `ajaxStop` from being triggered. This can be used to control various Ajax Events.

### 3.3.14 method

- default: `'GET'`
- type: `String`
- function: The HTTP method to use for the request (e.g. POST, GET, PUT).

### 3.3.15 success

- default: does not apply
- type: `Function`
- function: A function to be called if the request succeeds. The function gets passed three arguments: The data returned from the server, formatted according to the `dataType` parameter or the `dataFilter` callback function, if specified; a string describing the status; and the `jqXHR` (in jQuery 1.4.x, XMLHttpRequest) object.

### 3.3.16 timeout

- default: does not apply
- type: `Number`
- function: Set a timeout (in milliseconds) for the request. This will override any global timeout set with `$.ajaxSetup()`. The timeout period starts at the point the `$.ajax` call is made; if several other requests are in progress and the browser has no connections available, it is possible for a request to time out before it can be sent.

### 3.3.17 type

- default: `'GET'`
- type: `String`
- function: An alias for `method`. You should use type if you're using versions of jQuery prior to 1.9.0.

### 3.3.18 url

- default: `'current page'`
- type: `String`
- function: A string containing the URL to which the request is sent.

### 3.3.19 username

- default: does not apply
- type: `String`
- function: A username to be used with XMLHttpRequest in response to an HTTP access authentication request.

### 3.3.20 password

- default: does not apply
- type: `String`
- function: A password to be used with XMLHttpRequest in response to an HTTP access authentication request.

To be as precise as possible, information on AJAX options/settings is taken from the official website, where you can find even more by clicking here.

## 3.4 Where to use AJAX?

Ajax should be used anywhere in a web application where small amounts of information could be saved or retrieved from the server without posting back the entire pages. A good example of this is data validation on save actions. Another example would be to change the values in a drop down list-box based on other inputs, such as state and college list boxes. When the user selects a state, the college list box will repopulate with only colleges and universities in that state.

Another great example is when the client needs to save or retrieve session values from the server, based on a user preference such as the height, width or position of an object. Adjusting the width could make a callback to the server to set the session variable for the new width. This way, whenever the page is refreshed, the server can adjust the object's width based on this session variable. Otherwise, the object would go back to its initial default width.

Other features include text hints and autocomplete text boxes. The client types in a couple of letters and a list of all values that start with those letters appear below. A callback is made to a web service that will retrieve all values that begin with these characters. This is a fantastic feature that would be impossible without Ajax and is also part of the Ajax Control Toolkit.

Credits go to SegueTech for providing a great overview on AJAX use cases.

## 3.5 Conclusion

There are so much ways you can benefit from using AJAX that you barely have time to notice all. However, it is important to learn the basics and expand knowledge on the various options that we presented above. This way you'll know exactly when to use certain options to achieve data retrieval, on several conditions! AJAX can be as well used with php or other back-end programming languages, but here we focused on jQuery, where it gets its' most usable features and it is easier to implement.

**Note**: You can only try and see the results of this code in Internet Explorer because we used offline files to demonstrate AJAX, which other browser cannot handle or require extra set up.

## 3.6 Download

**Download** You can download the full source code of this example here: **jQuery AJAX**

# Chapter 4

# File Upload

The aim of this example is to give you the right knowledge about how you can achieve file upload with jQuery. Notice that this is not an easy task, and plugins are recommended to have non-surprising results.

In particular, there is one famous and very used jQuery Plugin for file upload made public to GitHub by **blueimp**. They created a file upload widget with multiple file selection, drag&drop support, progress bar, validation and preview images, audio and video for jQuery.

It supports cross-domain, chunked and resumable file uploads. Works with any server-side platform (Google App Engine, PHP, Python, Ruby on Rails, Java, etc.) that supports standard HTML form file uploads.

## 4.1 Plugin Demo and Features

### 4.1.1 Video Demo #1 - Single & Multiple Files Upload

Look at the demo in this video: jquery-fileupload-1.mp4

### 4.1.2 Video Demo #2 - Drag & Drop File Upload

Look at the demo in this video: jquery-fileupload-2.mp4

### 4.1.3 Features

- **Multiple file upload:** Allows to select multiple files at once and upload them simultaneously.
- **Drag & Drop support:** Allows to upload files by dragging them from your desktop or filemanager and dropping them on your browser window.
- **Upload progress bar:** Shows a progress bar indicating the upload progress for individual files and for all uploads combined.
- **Cancelable uploads:** Individual file uploads can be canceled to stop the upload progress.
- **Resumable uploads:** Aborted uploads can be resumed with browsers supporting the Blob API.
- **Chunked uploads:** Large files can be uploaded in smaller chunks with browsers supporting the Blob API.
- **Client-side image resizing:** Images can be automatically resized on client-side with browsers supporting the required JS APIs.
- **Preview images, audio and video:** A preview of image, audio and video files can be displayed before uploading with browsers supporting the required APIs.

- **No browser plugins (e.g. Adobe Flash) required:** The implementation is based on open standards like HTML5 and JavaScript and requires no additional browser plugins.
- **Graceful fallback for legacy browsers:** Uploads files via XMLHttpRequests if supported and uses iframes as fallback for legacy browsers.
- **HTML file upload form fallback:** Allows progressive enhancement by using a standard HTML file upload form as widget element.
- **Cross-site file uploads:** Supports uploading files to a different domain with cross-site XMLHttpRequests or iframe redirects.
- **Multiple plugin instances:** Allows to use multiple plugin instances on the same webpage.
- **Customizable and extensible:** Provides an API to set individual options and define callBack methods for various upload events.
- **Multipart and file contents stream uploads:** Files can be uploaded as standard "multipart/form-data" or file contents stream (HTTP PUT file upload).
- **Compatible with any server-side application platform:** Works with any server-side platform (PHP, Python, Ruby on Rails, Java, Node.js, Go etc.) that supports standard HTML form file uploads.

## 4.2 Basic Plugin Setup

### 4.2.1 HTML Setup

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>jQuery File Upload Example</title>
</head>
<body>
<input id="fileupload" type="file" name="files[]" data-url="server/php/" multiple>
<script src="//ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"></script>
<script src="js/vendor/jquery.ui.widget.js"></script>
<script src="js/jquery.iframe-transport.js"></script>
<script src="js/jquery.fileupload.js"></script>
<script>
$(function () {
    $('#fileupload').fileupload({
        dataType: 'json',
        done: function (e, data) {
            $.each(data.result.files, function (index, file) {
                $('').text(file.name).appendTo(document.body);
            });
        }
    });
});
</script>
</body>
</html>
```

**Response and data type**

The example above sets the dataType option to json, expecting the server to return a JSON response for each uploaded file. However it's also possible to handle HTML content types as response or any other dataType that you can handle in your done handler.

### 4.2.2 Display Upload Progress

The fileupload plugin triggers progress events for both individual uploads (progress) and all running uploads combined (progressall). Event handlers can be set via the event binding mechanism or as widget options.

```
$('#fileupload').fileupload({
    /* ... */
    progressall: function (e, data) {
        var progress = parseInt(data.loaded / data.total * 100, 10);
        $('#progress .bar').css(
            'width',
            progress + '%'
        );
    }
});
```

The previous code assumes a progress node with an inner element that displays the progress status via its width percentage:

```
    <div class="bar" style="width: 0%;">
</div>
```

The inner element should have a different background color than the container node, set via CSS and needs a height applied:

```
.bar {
    height: 18px;
    background: green;
}
```

### 4.2.3 Tie a file to an element node during the life cycle of an upload

Often, you will display a file to upload in an element node. This can be done in the **add** callback.

To be able to refer to the same element node in other callbacks related to the upload, you can make use of the **context** option (which is actually an option of jquery.ajax):

```
$(function () {
    $('#fileupload').fileupload({
        dataType: 'json',
        add: function (e, data) {
            data.context = $('<p />').text('Uploading...').appendTo(document.body);
            data.submit();
        },
        done: function (e, data) {
            data.context.text('Upload finished.');
        }
    });
});
```

### 4.2.4 Start Uploads with a button click

Based on the previous example, it's possible to start uploads on the click of a button instead of automatically:

```
$(function () {
    $('#fileupload').fileupload({
        dataType: 'json',
        add: function (e, data) {
            data.context = $('<button />').text('Upload')
                .appendTo(document.body)
                .click(function () {
```

```
                    data.context = $('<p />').text('Uploading...').replaceAll($(this));
                    data.submit();
                });
        },
        done: function (e, data) {
            data.context.text('Upload finished.');
        }
    });
});
```

## 4.3 Requirements

### 4.3.1 Mandatory Requirements

- jQuery v. 1.6+
- jQuery UI widget factory v. 1.9+ (included)
- jQuery Iframe Transport plugin (included)

The jQuery UI widget factory is a requirement for the basic File Upload plugin, but very lightweight without any other dependencies from the jQuery UI suite. The jQuery Iframe Transport is required for browsers without XHR file upload support.

### 4.3.2 Optional Requirements

- JavaScript Templates engine v. 2.5.4+
- JavaScript Load Image library v. 1.13.0+
- JavaScript Canvas to Blob polyfill v. 2.1.1+
- blueimp Gallery v. 2.15.1+
- Bootstrap v. 3.2.0+
- Glyphicons

The **JavaScript Templates engine** is used to render the selected and uploaded files for the Basic Plus UI and jQuery UI.

The **JavaScript Load Image library** and JavaScript Canvas to Blob polyfill are required for the image previews and resizing functionality.

The **blueimp Gallery** is used to display the uploaded images in a lightbox.

The user interface of all versions except the jQuery UI version is built with **Bootstrap** and icons from **Glyphicons**.

## 4.4 Browsers

### 4.4.1 Desktop Browsers

The File Upload plugin is regularly tested with the latest browser versions and supports the following minimal versions:

- Google Chrome
- Apple Safari 4.0+

- Mozilla Firefox 3.0+
- Opera 11.0+
- Microsoft Internet Explorer 6.0+
- Microsoft Edge

### 4.4.2 Mobile Browsers

The File Upload plugin has been tested with and supports the following mobile browsers:

- Apple Safari on iOS 6.0+
- Google Chrome on iOS 6.0+
- Google Chrome on Android 4.0+
- Default Browser on Android 2.3+
- Opera Mobile 12.0+

## 4.5 Conclusion

To conclude, file upload with jQuery can be easily adapted to your websites using the plugin we presented and its' features. You can find this plugin on GitHub following the link https://github.com/blueimp/jQuery-File-Upload. It also has detailed information on the plugin useage and a live demo. However, if you feel that you are searching for something else Kendo UI has another solution for file uploads that you can find here.

## 4.6 Download

**Download** You can download the full source code of this example here: **jQuery File Upload**

# Chapter 5

# Drag and Drop

The aim of this example is to explain and use drag and drop functionality with jQuery. Basically, in jQuery you can enable draggable functionality on any DOM element and move the draggable object by clicking on it with the mouse and dragging it anywhere within the viewport. Dragging and dropping can be a very intuitive way for users to interact with your site or web app. People often use drag-and-drop for things like:

- Moving email messages into folders
- Reordering lists of items
- Moving objects in games around, such as cards and puzzle pieces

## 5.1 Basic Setup & Application

To begin, create and new HTML document and add the following basic syntax inside:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
        <title>jQuery Drag &amp;amp; Drop Example</title>

    <!-- LINKS SECTION  -->
    <link rel="stylesheet" href="jquery-ui.css">
    <script src="jquery-1.11.3.min.js"></script>
    <script src="jquery-ui.js"></script>
    <link rel="stylesheet" href="style.css">
</head>
<body>

<!-- STYLE SECTION -->
<style type="text/css">

</style>

<!-- HTML SECTION  -->

<!-- JAVASCRIPT SECTION  -->

</script>

</body>
</html>
```

Don't worry about the link files, you'll find them attached in your final source download at the end of the article.

Now let's see a basic drag and drop example. First, create a new element in the HTML section and give it a class name.

```
<!-- HTML SECTION  -->
Drag me!
```

Next, to make it more clear, give this element some styling:

```
<!-- STYLE SECTION -->
<style type="text/css">
.dragme {
        border-radius: 0.5em;
        width: 12em;
        height: 8em;
        padding: 1em;
        font-family: "Montserrat", "Arial";
        background-color: #00C5CD;
        color: white;
    }
</style>
```

Next, in your Javascript section, create a new function which finds the `.dragme` class and applies the `.draggable()` method to this class:

```
<!-- JAVASCRIPT SECTION  -->
<script type="text/javascript">
  $(function() {
    $('.dragme').draggable();
  });
</script>
```

The result would be a draggable element like this.

## 5.2 Customized Draggable Elements

In this section, we'll have a look at how much we can customize draggable elements to fit our needs.

### 5.2.1 Constraining Movement

Constrain the movement of each draggable by defining the boundaries of the draggable area. Set the axis option to limit the draggable's path to the x- or y-axis, or use the containment option to specify a parent DOM element or a jQuery selector, like *document*. Let's first use the `containment` to limit the area to a parent div:

```
<!-- HTML SECTION  -->

    <div class="dragme">Drag me!
</div>
```

Let's give the parent div a little styling:

```
<!-- STYLE SECTION -->
<style type="text/css">
.drag-parent {
    border: 0.1em solid #BA064E;
    width: 45em;
    height: 20em;
}
</style>
```

And now, let's add the `containment` option to the `draggable()` method:

```
<!-- JAVASCRIPT SECTION  -->
<script type="text/javascript">
  $(function() {
    $('.dragme').draggable({ containment: ".drag-parent" });
  });
</script>
```

This would result in a limited draggable area.

Check out the functionality here.

Now let's constrain draggable area by axis, x or y. To do this, add the `axis` option to the `.draggable()` method and give it a value of `'x'` or `'y'` depending on what you are looking for.

```
<!-- HTML SECTION  -->
    Drag me by x!
    Drag me by y!
```

```
<!-- JAVASCRIPT SECTION  -->
<script type="text/javascript">
  $(function() {
    $('.dragme-x').draggable({ axis: "x" });
    $('.dragme-y').draggable({ axis: "y" });
  });
</script>
```

Check out the functionality here.

### 5.2.2 Cursor Styles over Draggable Element

Position the cursor while dragging the object. By default the cursor appears in the center of the dragged object; use the `cursorAt` option to specify another location relative to the draggable (specify a pixel value from the `top`, `right`, `bottom`, and/or `left`). Customize the cursor's appearance by supplying the cursor option with a valid CSS cursor value: default, move, pointer, crosshair, etc.

```
<!-- HTML SECTION  -->
    Drag me by x!
    Drag me by y!
```

```
<!-- JAVASCRIPT SECTION  -->
<script type="text/javascript">
$(function() {
$('.dragme-crosshair').draggable({ cursor: "crosshair", cursorAt: { top: 80, left: 120 } }) ↩
    ;
$('.dragme-move').draggable({ cursor: "move", cursorAt: { top: 90, left: 100 } });
});
</script>
```

The result would be a customized cursor while dragging our element. Check out the functionality here.

### 5.2.3 Revert Draggable Element Position

You can actually return the draggable (or it's helper) to its original location when dragging stops with the boolean `revert` option.

```
<!-- JAVASCRIPT SECTION  -->
<script type="text/javascript">
  $(function() {
    $( '.dragme' ).draggable({ revert: true });
  });
</script>
[source,java]
```

Check out the functionality here.

Another useful option here would be `helper` with the value `clone`. That would create a clone of the element while dragging and still show the original element right in its inital position:

```
  $(function() {
    $( '.dragme' ).draggable({ revert: true, helper: "clone" });
  });
[source,java]
```

Check out the functionality here.

### 5.2.4 Snap to Element or Grid

Snap the draggable to the inner or outer boundaries of a DOM element. Use the snap or snapMode (inner, outer, both). Or snap the draggable to a grid. Set the dimensions of grid cells (height and width in pixels) with the grid option.

```
<!-- HTML SECTION  -->

    <div class="dragme dragme-normal">I snap to all other draggable elements!
    Drag me to my parent!
    Drag me to my grid!
</div>
```

```
<!-- JAVASCRIPT SECTION  -->
<script type="text/javascript">
  $(function() {
    $('.dragme-normal').draggable({ snap: true });
    $('.dragme-parent').draggable({ snap: '.drag-parent' });
    $('.dragme-grid').draggable({ grid: [ 50, 50 ] });
  });
</script>
```

Check out the functionality here.

## 5.3 An Advanced Approach

In a more professional and complete demo that you can find here, you can make this ultimate by adding or removing congtent (classes) on mouseover and mouseleave to complete the whole user experience for the modern web. The demo includes your source code for this. Check out the functionality here.

## 5.4 Conclusion

Drag-and-drop with JavaScript used to be very hard to do - in fact, getting a decent cross-browser version working was next to impossible. However, with modern browsers and a smattering of jQuery, drag-and-drop is now a piece of cake! Dragging has a lot to customize and improve to make your UX perfect. Try out and see how you can get creative and productive!

## 5.5 Download

**Download** You can download the full source code of this example here: **jQuery Drag & Drop**

# Chapter 6

# UI Autocomplete

In this example, we're going through a very useful widget of jQuery, `autocomplete()`.

Autocomplete enables users to quickly find and select from a pre-populated list of values as they type, leveraging searching and filtering. Any field that can receive input can be converted into an Autocomplete, namely, `<input>` elements, `<textarea>` elements, and elements with the contenteditable attribute.

By giving an Autocomplete field focus or entering something into it, the plugin starts searching for entries that match and displays a list of values to choose from. By entering more characters, the user can filter down the list to better matches.

This can be used to choose previously selected values, such as entering tags for articles or entering email addresses from an address book. Autocomplete can also be used to populate associated information, such as entering a city name and getting the zip code.

## 6.1 Document Setup

In order to have a good start, after creating a new HTML document, add the following basic syntax and jQuery links into it:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>jQuery UI Autocomplete Example</title>

<!-- LINKS SECTION  -->
  <link href="http://ajax.googleapis.com/ajax/libs/jqueryui/1.8/themes/base/jquery-ui.css"  ↩
      rel="stylesheet" type="text/css"/>
  <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js"></script>
  <script src="http://ajax.googleapis.com/ajax/libs/jqueryui/1.8/jquery-ui.min.js"></script ↩
      >
  <link href="style.css" rel="stylesheet" type="text/css"/>

<!-- JAVASCRIPT SECTION  -->

</head>
<body>
<!-- HTML SECTION  -->

</body>
</html>
```

Now that we've included the most important links like the jquery-ui and jquery javascript files, we're ready to create a basic autocomplete field.

## 6.2 Basic Autocomplete Input Field

So basically, what we're doing here is creating a variable inside a function, which will hold the actual suggested (autocomplete) words, and then use that source to populate the filtered list as we type in the input field. The schema below shows how we're structuring our code to do this:

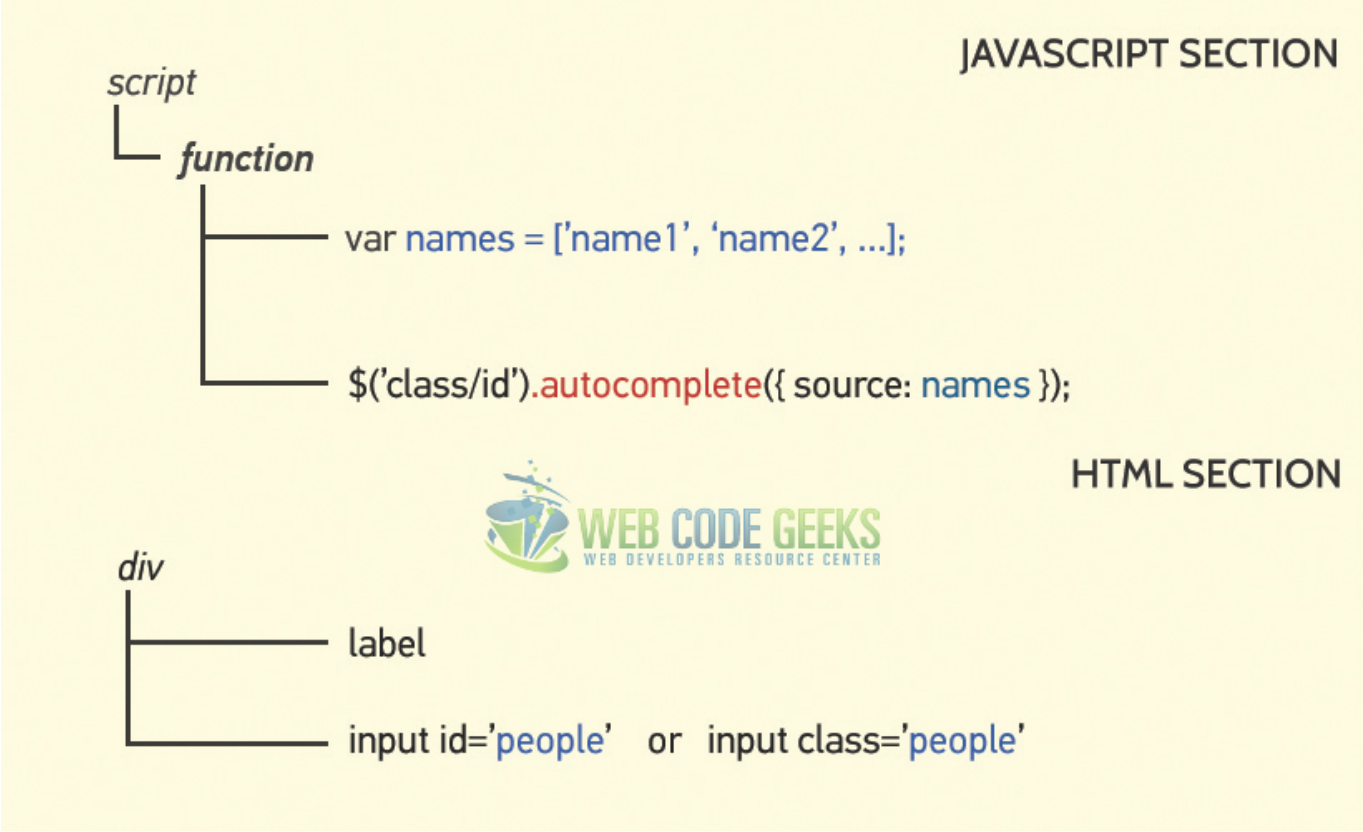


Figure 6.1: Our Code Structure

That's it. Let's go ahead and do it:

```
<!-- HTML SECTION  -->

    <!-- this class could be anything you want -->

  <label for="people">Tags: </label>

  <input class="people">
```

```
<!-- JAVASCRIPT SECTION  -->

<script>

  $(function() {

    var names = [

      "Alban",
      "Andy",
      "Ajax",
```

```
      "Bob",
      "Cody",
      "Chloe",
      "Camela",
      "Charlotte",
      "Ciara",
      "Ella",
      "Fabio",
      "George",
      "Helen",
      "Juliet",
      "James",
      "Lory",
      "Patricia",
      "Peter",
      "Roxanna",
      "Randi",
      "Selena",
      "Sara"
    ];

    $(".people").autocomplete({
    /*refer to the same id that input has*/
      source: names
    /*set the source name that you gave your array variable*/
    });
  });
  </script>
```

Trying this out in the browser would result in getting constant suggestions as we type.

Check out the functionality here.

## 6.3 Autocomplete Options

Autocomplete has some interesting options that will be helpful when using the autocomplete method. Below, we'll have a look at some of them.

### 6.3.1 AppendTo

Add a new element in HTML, that will be the element where filtered words will be added.

```
<!-- HTML SECTION  -->

  <label for="people">Tags: </label>
  <input class="people">
  <p class="para"> <!-- added this line -->
```

Now initialize the autocomplete with the `appendTo` option specified:

```
$(".people").autocomplete({
    appendTo: ".para"
});
```

Get or set the `appendTo` option, after initialization:

```
// Getter
var appendTo = $( ".people" ).autocomplete( "option", "appendTo" );
```

```
// Setter
$( ".people" ).autocomplete( "option", "appendTo", ".para" );
```

The result would show names getting appended to the paragraph as soon as they are searched by a letter.

Look at the results in this video: autocomplete-3

### 6.3.2 Delay

The delay in milliseconds between when a keystroke occurs and when a search is performed. A zero-delay makes sense for local data (more responsive), but can produce a lot of load for remote data, while being less responsive.

Initialize the autocomplete with the `delay` option specified:

```
$( ".selector" ).autocomplete({
  delay: 500
});
```

Get or set the `delay` option, after initialization:

```
// Getter
var delay = $( ".selector" ).autocomplete( "option", "delay" );

// Setter
$( ".selector" ).autocomplete( "option", "delay", 500 );
```

Check out the functionality here.

### 6.3.3 Disabled

Disables the autocomplete if set to `true`.

Initialize the autocomplete with the `disabled` option specified:

```
$( ".selector" ).autocomplete({
  disabled: true
});
```

Get or set the `disabled` option, after initialization:

```
// Getter
var disabled = $( ".selector" ).autocomplete( "option", "disabled" );

// Setter
$( ".selector" ).autocomplete( "option", "disabled", true );
```

Check out the functionality here.

### 6.3.4 minLength

The minimum number of characters a user must type before a search is performed. Zero is useful for local data with just a few items, but a higher value should be used when a single character search could match a few thousand items.

Initialize the autocomplete with the `minLength` option specified:

```
$( ".selector" ).autocomplete({
  minLength: 2
});
```

Get or set the `minLength` option, after initialization:

```
// Getter
var minLength = $( ".selector" ).autocomplete( "option", "minLength" );

// Setter
$( ".selector" ).autocomplete( "option", "minLength", 2 );
```

As a result, autocompletion will only start after typing the second character in the input field.

Check out the functionality here.

### 6.3.5 Source

Defines the data to use, must be specified. It may be an Array, a String or a Function.

Initialize the autocomplete with the `source` option specified:

```
$( ".selector" ).autocomplete({
  source: [ "c++", "java", "php", "coldfusion", "javascript", "asp", "ruby" ]
});
```

Get or set the `source` option, after initialization:

```
// Getter
var source = $( ".selector" ).autocomplete( "option", "source" );

// Setter
$( ".selector" ).autocomplete( "option", "source", [ "c++", "java", "php", "coldfusion", " ↩
    javascript", "asp", "ruby" ] );
```

Notice that we already used the `source` option as the most basic option, but we used an array of names apart, instead of inline declaration.

## 6.4 Conclusion

The autocomplete widget is a useful tool to consider when dealing with input fields, it gives the user a suggestion (or maybe a hint) on what the input can be filled with. As we saw here, it is highly customizable and helps you optimize it. For more information on the `autocomplete` widget, feel free to use the official jQuery UI wesbite and specifically this topic.

## 6.5 Download

**Download** You can download the full source code of this example here: **jQuery UI Autocomplete**

# Chapter 7

# CSS Background Image

In this example, we'll learn how to use jQuery to add CSS properties to HTML elements and specifically how to add backgrounds like colors or images using the `.css()` method.

The `.css()` method is a convenient way to get a computed style property from the first matched element, especially in light of the different ways browsers access most of those properties (the `getComputedStyle()` method in standards-based browsers versus the `currentStyle` and `runtimeStyle` properties in Internet Explorer) and the different terms browsers use for certain properties.

For example, Internet Explorer's DOM implementation refers to the `float` property as `styleFloat`, while W3C standards-compliant browsers refer to it as `cssFloat`. For consistency, you can simply use "`float`", and jQuery will translate it to the correct value for each browser.

## 7.1 Basic Setup

### 7.1.1 Initial Document Setup

To begin, create a new HTML document and add the following sections and links:

```
<!DOCTYPE html>
<html>
<head>
        <title>jQuery CSS Background Image Example</title>
</head>
<body>
<!-- STYLE SECTION  -->

<!-- HTML SECTION  -->

<!-- JAVASCRIPT SECTION  -->
<script src="jquery-1.11.3.min.js"></script>

<script type="text/javascript">
// our jQuery code goes here
</script>

</body>
</html>
```

### 7.1.2 Understanding the .css() method

`.css( propertyName )` - Get the computed style properties for the first element in the set of matched elements. propertyName will be a string containing the name of a CSS property. Look at the following example:

```
<!-- HTML SECTION  -->
<span class="result"></span>  <!-- show the computed results here -->
Click to show two of my css properties.
```

```
<!-- STYLE SECTION  -->
<style type="text/css">
.content {
    width: 20em;
    height: 10em;
    margin: 1em;
    background-color: #FB2A59;
    text-align: center;
    line-height: 10em;
    color: white;
    border-radius: 0.5em;
}
</style>
```

```
<!-- JAVASCRIPT SECTION  -->
<script src="jquery-1.11.3.min.js"></script>

<script type="text/javascript">
    $(function (){
        $('.content').click(function(){ /*results will be shown on click*/
            var width = $(this).css("width");   /*store the width in a variable*/
            var height = $(this).css("height"); /*store the height in a variable*/
            /*concatinate several properties and attach them to some other element*/
            $('.result').html("Width: " + width + "<br>" + "Height: " + height);
        });
    })
</script>
```


Figure 7.1: Using .css() - Single Property

But you can use the .css() method with multiple properties inside: `.css( propertyNames )` where `propertyNames` would represent an array of one or more CSS properties. Modifying the example above, we'd get:

```
<!-- JAVASCRIPT SECTION  -->
<script src="jquery-1.11.3.min.js"></script>

<script type="text/javascript">
    $(function (){
        var html = [ "The clicked div has the following styles:" ];
        $('.content').click(function(){ /*results will be shown on click*/
            /*store the css properties array in a variable*/
            var properties = $(this).css(["width", "height", "background-color", "color"]);
            /*concatinate several properties and attach them to some other element*/
              $.each( properties, function( prop, value ) {
                html.push( prop + ": " + value );
              });
              $( ".result" ).html( html.join( "<br>" ) );
        });
    })
</script>
```


Figure 7.2: Using .css() - Multiple Properties

## 7.2 Background Image using .css()

Now let's try to add a background color and then a background image in a content box. The easiest way to do this is to refer to the element you want to give a background color and then use `.css('background-color', '#eee')` like so:

```
<!-- JAVASCRIPT SECTION  -->
<script type="text/javascript">
    $(function (){
        $('.content').css('background-color', '#51326F');
    })
</script>
```

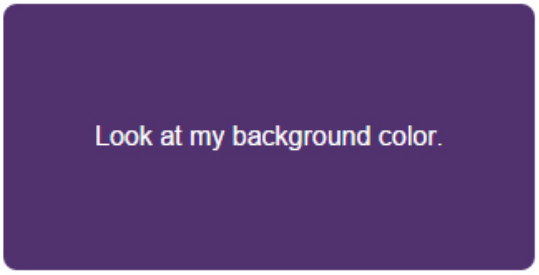


Figure 7.3: jQuery Background Image

In a similar manner, we can use the syntax `.css('background-image', 'url(image.jpg)')` to add a background image like so:

```
<!-- JAVASCRIPT SECTION  -->
<script type="text/javascript">
    $(function (){
        $('.content').css('background-image', 'url(bg.jpg)');
    })
</script>
```

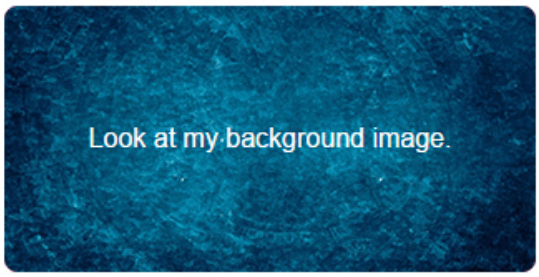


Figure 7.4: jQuery Background Image

You can choose to show the background image we just set with jQuery only on click. You can do that like this:

```
<!-- JAVASCRIPT SECTION  -->
<script type="text/javascript">
    $('.content1').click(function(){
            $(this).css('background-image','url(bg1.jpg)');
            $(this).find('p').hide();
            $(this).html("Nice Job, User!");
    });
</script>
```

Check out the functionality here.

## 7.3 Conclusion

To conclude, changing the background of an element with jQuery becomes really useful and necessary when you want to trigger these events on certain actions taken by the user or when you want to create functions to manipulate the background for some reason like animation ect. At all times, keep in mind the basic syntax of .css() method as it is an essential jQuery method to be used to set or change CSS properties.

## 7.4 Download

**Download** You can download the full source code of this example here: **jQuery CSS Background Image**

# Chapter 8

# Disable Button

The aim of this example is to show you how to enable/disable a button using the famous jQuery library of Javascript.

This is a pretty simple task but very useful on certain cases like when you want to submit a form and disable the button that did so, or just because a button is part of a conditional statement and it should be disabled if one of the conditions is true/false. Let's have a further look into it.

## 8.1 Basic Setup

To start fresh, just create a new HTML document with its basic syntax inside and link your jQuery file inside like so:

```
<!DOCTYPE html>
<html>
<head>
        <title>Basic Example</title>
</head>
<body>

<!-- STYLE  SECTION -->
<style type="text/css">

</style>

<!-- HTML SECTION  -->


<!-- JAVASCRIPT SECTION  -->
<script src="jquery-1.11.3.min.js"></script>

<script type="text/javascript">

</script>

</body>
</html>
```

In order to continue with jQuery, let's first add a new `button` in HTML like so:

```
<!-- HTML SECTION  -->
<button>Send Details</button>
```

## 8.2 Disabling a Button with jQuery

There are several cases and ways you can and want to disable a button, so here are the most important ones!

### 8.2.1 Disabled as an Initial State of the Button

There might be cases you want to set a button as disabled by default since the opening of the page. With jQuery, you can do that using the `.attr` method.

```
<!-- JAVASCRIPT SECTION  -->
<script src="jquery-1.11.3.min.js"></script>

<script type="text/javascript">
    $(document).ready(function(){   /*execute code after the page has been loaded*/
        /*reference the button and change its disabled attribute to 'disabled'*/
        $('button').attr('disabled', 'disabled');
    });
</script>
```

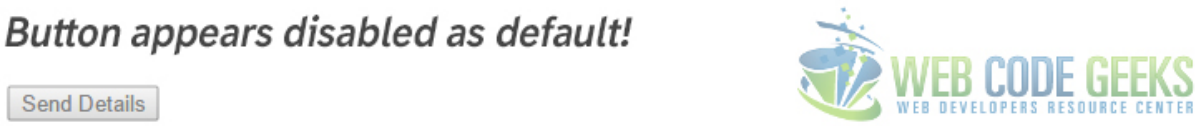


Figure 8.1: Disabling a Button as an Initial State

### 8.2.2 Disabling a Button on Click

What if you want to disable a button as soon as it is clicked by the user. Well, you can do that by adding a function which will be executed on any click on the button.

```
<script type="text/javascript">
    $(document).ready(function(){
        // reference a button and execute a 'function' when it is clicked
        $('button').click(function(){
            // reference 'this' (button), and change the disabled attribute to 'disabled'
            $(this).attr('disabled', 'disabled');
        })
    });
</script>
```

Now clicking on that button will disable it.


Figure 8.2: Disabling a Button on Click

### 8.2.3 Disabling a Button after Form Submission

A useful case when it would be obvious to disable a button is when a form submission button is clicked. First, add some lines to create a form in HTML:

```
<form>
    <input type="text" placeholder="Name">
    <input type="text" placeholder="Age">
    <button>Send Details</button>
</form>
```

To disable the button on form submission, first we reference the form and use the .submit event listener to execute a function we define:

```
<script type="text/javascript">
    $(document).ready(function(){
    // reference the form element and watch for 'form' submission event
        $('form').submit(function(e){
        // prevent the default browser behaviour on this case
        e.preventDefault();
        // reference 'this' (the form) then find the 'button'
        // change its disabled attribute to 'disabled'
       $(this).find('button').attr('disabled', 'disabled');
    }); });
</script>
```


Figure 8.3: Disabling a Button after Form Submission

In another scenarion, when our button would be represented as an input element of `type="submit"` like this:

```
<input type="submit">
```

disabling it in jQuery would look like this:

```
// reference children of the 'form' element which have an 'input' with a type of submit
// disable that input using the disabled value of the disabled attribute
$(this).children('input[type=submit]').attr('disabled', 'disabled');
```

The idea is the same, only the way we define the button is changed.

## 8.3 Conclusion

Disabling a button is just a normal action to take whenever you need to. With jQuery, this is easy and short in code. However, do remember to reference the right elements/classes when on larger documents and notice to differentiate buttons using classes (therefore, referencing classes) in case you don't won't to apply the *disabled* state to all of them.

## 8.4 Download

**Download** You can download the full source code of this example here: **jQuery Disable Button Example**