# AJAX

## *Asynchronous JavaScript And ~~XML~~ JSON*

**AJAX** was first introduced into JavaScript in **1999** and has since become a **fundamental tool in modern web applications.**

**AJAX** allows us to send and retrieve data from the web server **without refreshing the page.**

**AJAX** originally was used to send and retrieve **XML** documents, but is used almost exclusively with **JSON** documents today.

**AJAX** is a good fit for the following functionality:

- Checking if a username / email address is in use by another user
- Providing autocomplete functionality for search boxes
- Spell Checking
- Progress Bars for upload pages
- Contact Us forms
- Likes
- 5-Star Ratings
- Comments
- Online Chat / Instant Messaging
- Continuous Scrolling
- *And many more things...*

Using **AJAX** to perform these operations also allows us to use **animations** to improve the overall feel of the application.

Using **AJAX** often requires us to bridge the world of **C# & Razor which is run on the server, with the world of Javascript which is run on the browser.** This can lead to some interesting cases where C# and JavaScript are intermingled.

**Further Reading**
**https://en.wikipedia.org/wiki/Ajax_(programming)**
**https://codeburst.io/modern-state-of-ajax-f2ac91d721b1**
**https://medium.com/letsboot/basics-using-ajax-with-fetch-api-b2218b0b9691**

**ASP.NET MVC Examples**
**https://www.aspsnippets.com/Articles/ASPNet-MVC-jQuery-AJAX-and-JSON-Example.aspx**
**http://www.codedigest.com/posts/39/using-jquery-ajax-methods-to-get-json-result-from-controller-in-aspnet-mvc**

# XHR and jQuery

The **XMLHttpRequest (XHR)** object is used to make asynchronous HTTP requests in Vanilla JavaScript.

> "Use XMLHttpRequest (XHR) objects to interact with servers. You can retrieve data from a URL without having to do a full page refresh. This enables a Web page to update just part of a page without disrupting what the user is doing. XMLHttpRequest is used heavily in AJAX programming.
> Despite its name, XMLHttpRequest can be used to retrieve any type of data, not just XML."
>
> **MDN web docs**
> **https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest**

To avoid browser compatibility problems, we generally use **jQuery** to perform asynchronous HTTP requests.

> "**The $.ajax() function underlies all Ajax requests sent by jQuery.** It is often unnecessary to directly call this function, as several higher-level alternatives like **$.get(), $.post(), and .load()** are available and are easier to use. If less common options are required, though, $.ajax() can be used more flexibly."
>
> "The jQuery XMLHttpRequest **(jqXHR)** object returned by **$.ajax()** is a superset (or subclass) of the browser's native XMLHttpRequest object. For example, it contains responseText and responseXML properties, as well as a getResponseHeader() method."
>
> "The **jqXHR** objects returned by **$.ajax()** implement the **Promise** interface, giving them all the properties, methods, and behavior of a **Promise** (see Deferred Object for more information). These methods take a function argument that is called when the $.ajax() request terminates. [...] Available Promise methods of the jqXHR object include:
>
> - **jqXHR.done(function( data, textStatus, jqXHR ) {})**
>   Callback is executed when request is successful.
>   https://api.jquery.com/deferred.done/
>
> - **jqXHR.fail(function( jqXHR, textStatus, errorThrown ) {})**
>   Callback is executed when request is unsuccessful.
>   https://api.jquery.com/deferred.fail/
>
> **jQuery api docs**
> https://api.jquery.com/jquery.ajax/

# $.get()

$.get() is a shorthand AJAX function, which is equivalent to:

```
$.ajax({
  type: "GET"
  url: url,
  data: data,
  success: success,
  dataType: dataType
});
```

## Examples

Request the test.php page, but ignore the return results.

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

Request the test.php page and send some additional data along (while still ignoring the return results).

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

Pass arrays of data to the server (while still ignoring the return results).

```
$.get( "test.php", { "choices[]": ["Jon", "Susan"] } );
```

Alert the results from requesting test.php

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

Alert the results from requesting test.php with an additional payload of data.

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

Get the test.php page contents, which has been returned in json format, and add it to the page.

```
$.get( "test.php" )
  .done(function (data) {
    $( "body" )
      .append( "Name: " + data.name ) // John
      .append( "Time: " + data.time ); // 2pm
  });
```

https://api.jquery.com/jquery.get/

# $.post()

$.post() is a shorthand AJAX function, which is equivalent to:

```
$.ajax({
  type: "POST"
  url: url,
  data: data,
  success: success,
  dataType: dataType
});
```

## Examples

Alert the results from requesting test.php

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

Alert the results from requesting test.php with an additional payload of data.

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

Post to the test.php page and get content which has been returned in json format.

```
$.post( "test.php" )
  .done(function (data) {
    $( "body" )
      .append( "Name: " + data.name ) // John
      .append( "Time: " + data.time ); // 2pm
  });
```

Post a form using Ajax and put results in a div.

```
See bottom of https://api.jquery.com/jquery.post/
```

# .load()

.load() can be used in some simple cases to dynamically fetch fragments of HTML from the server.
However in most cases you will want to send and receive JSON data from the server using $.get() and $.post()

**Example**

```
$( "#result" ).load( "ajax/test.html", function () {
  alert( "Load was performed." );
});
```

# jQuery api docs

Read more about how to send AJAX requests with jQuery on the following pages.

https://api.jquery.com/jquery.ajax/
https://api.jquery.com/jquery.get/
https://api.jquery.com/jquery.post/
https://api.jquery.com/load/
https://api.jquery.com/category/deferred-object/
https://api.jquery.com/deferred.done/
https://api.jquery.com/deferred.fail/

# Razor Layout Sections

Using AJAX often requires to create page specific JavaScript scripts.

If we simply include these at the end of the view, they will be inserted before jQuery in the layout. Which means that they will not work as intended.

To include page specific scripts after jQuery is loaded we need to use **Razor Layout Sections.**

> **You can read more about using sections in Razor here:**
> **https://weblogs.asp.net/scottgu/asp-net-mvc-3-layouts-and-sections-with-razor**

In the **layout file** add the following line at the end of the body, after all your other scripts:

```
@RenderSection("Scripts", required: false)
```

In the **view file** add the following lines at the end of the body:

```
@section Scripts {
  <script>
    // page specific javascript goes here
  </script>
}
```

This will allow you to write custom scripts for each view.

It will also allow you to mix C#, Razor, and JavaScript code.

**Remember C# runs on the server, JavaScript runs in the browser.**

# Returning JSON Data from an Action Method

We are already familiar with returning HTML from an Action Method like so:

```
public ActionResult View(int productId) {
  Product p = _db.Products.SingleOrDefault(x => x.ProductId == productId);
  return View("View", p);
}
```

To support AJAX requests we often need to return data as JSON instead.

```
public ActionResult View(int productId) {
  Product p = _db.Products.SingleOrDefault(x => x.ProductId == productId);
  return Json(p, JsonRequestBehavior.AllowGet);
}
```

To avoid circular references on the navigation properties, we often need to add the **[ScriptIgnore]** attribute to navigation properties.

```
[Table("Products")]
public class Product {
  [Key]
  public int ProductId { get; set; }
  [Required]
  public string ProductName { get; set; }
  public int? CategoryId { get; set; }

  [ScriptIgnore]
  public virtual Category Category { get; set; }
  [ScriptIgnore]
  public virtual IList<OrderLines> OrderLines { get; set; }
}
```

**Controller.Json Method**
https://docs.microsoft.com/en-us/dotnet/api/system.web.mvc.controller.json?view=aspnet-mvc-5.2

**ScriptIgnoreAttribute Class**
https://docs.microsoft.com/en-us/dotnet/api/system.web.script.serialization.scriptignoreattribute?view=netframework-4.8

# JSON

## *JavaScript Object Notation*

**JSON** is simple and lightweight format for storing and transferring data. It started gaining traction in the early 2000s and quickly overtook XML as the primary data format used on the web.

**JSON** is based on a subset of the JavaScript Programming language, but is a language independent data format. Due to its simplicity and popularity there are JSON libraries for all major programming languages now.

**{ } is used to define Objects. [ ] is used to define Arrays.**

**Objects and Arrays can be nested inside of each other.**

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    { "type": "home", "number": "212 555-1234" },
    { "type": "office", "number": "646 555-4567" },
    { "type": "mobile", "number": "123 456-7890" }
  ],
  "children": [],
  "spouse": null
}
```

| | |
|---|---|
| **Read the official JSON specification here.**<br>http://www.json.org/ | **Wikipedia Article**<br>https://en.wikipedia.org/wiki/JSON |

## Web Services / API Endpoints

A **Web Service** is a service offered by one application to another application, communicating with each other via the internet.

**Action methods that return JSON data** can be referred to as **Web Services** or **API Endpoints,** since they return unformatted data that is used by code rather shown directly to the user.

https://en.wikipedia.org/wiki/Web_service

## Web APIs

**Application Programming Interfaces** or **APIs** have been around since the early days of programming.

Originally, an **API** was just a set of functions.
Today, an **API** is generally defined as a set of API Endpoints.

Each **Web API Endpoint** has a URL, an HTTP Method (such as GET, POST, PUT, or DELETE), a set of parameters it accepts, and return type. *(A lot like a method or function.)*

https://en.wikipedia.org/wiki/Application_programming_interface
https://en.wikipedia.org/wiki/Web_API

## RESTful APIs

RESTful APIs map CRUD functionality to different HTTP Methods.

```
GET /product
Return all products.

GET /product/5
Return product 5.

POST /product
Insert a new product.

PUT /product/5
Update product 5.

DELETE /product/5
Delete product 5.
```

https://restfulapi.net/
https://restfulapi.net/rest-put-vs-post/
https://en.wikipedia.org/wiki/Representational_state_transfer

# Postman

https://www.getpostman.com/

Postman is a popular tool to test Web APIs. It allows you to send HTTP Requests with any method, any parameters, and any headers you need.