jQuery.ajax()

jQuery.ajax(url [, settings]) ***Returns: jqXHR (http://api.jquery.com/Types/#jqXHR)***

Description: *Perform an asynchronous HTTP (Ajax) request.*

jQuery.ajax(url [, settings]) **version added: 1.5 (/category/version/1.5/)**

url
Type: String (http://api.jquery.com/Types/#String)
A string containing the URL to which the request is sent.

settings
Type: PlainObject (http://api.jquery.com/Types/#PlainObject)
A set of key/value pairs that configure the Ajax request. All settings are optional. A default can be set for any option with $.ajaxSetup() (/jQuery.ajaxSetup/). See jQuery.ajax(settings) below for a complete list of all settings.

jQuery.ajax([settings]) **version added: 1.0 (/category/version/1.0/)**

settings
Type: PlainObject (http://api.jquery.com/Types/#PlainObject)
A set of key/value pairs that configure the Ajax request. All settings are optional. A default can be set for any option with $.ajaxSetup() (/jQuery.ajaxSetup/).

accepts (default: `depends on DataType`)
Type: PlainObject (http://api.jquery.com/Types/#PlainObject)
A set of key/value pairs that map a given `dataType` to its MIME type, which gets sent in the `Accept` request header. This header tells the server what kind of response it will accept in return. For example, the following defines a custom type `mycustomtype` to be sent with the request:

```
$.ajax({
  accepts: {
    mycustomtype: 'application/x-some-custom-type'
  },

  // Instructions for how to deserialize a `mycustomtype
  converters: {
    'text mycustomtype': function(result) {
      // Do Stuff
      return newresult;
    }
  },

  // Expect a `mycustomtype` back from server
  dataType: 'mycustomtype'
});
```

Note: You will need to specify a complementary entry for this type in `converters` for this to work properly.

async (default: `true`)

Type: Boolean (http://api.jquery.com/Types/#Boolean)
By default, all requests are sent asynchronously (i.e. this is set to `true` by default). If you need synchronous requests, set this option to `false`. Cross-domain requests and `dataType: "jsonp"` requests do not support synchronous operation. Note that synchronous requests may temporarily lock the browser, disabling any actions while the request is active. **As of jQuery 1.8**, the use of `async: false` with jqXHR (`$.Deferred`) is deprecated; you must use the success/error/complete callback options instead of the corresponding methods of the jqXHR object such as `jqXHR.done()`.

---

beforeSend
Type: Function (http://api.jquery.com/Types/#Function)(jqXHR (http://api.jquery.com/Types/#jqXHR) jqXHR, PlainObject (http://api.jquery.com/Types/#PlainObject) settings)
A pre-request callback function that can be used to modify the jqXHR (in jQuery 1.4.x, XMLHTTPRequest) object before it is sent. Use this to set custom headers, etc. The jqXHR and settings objects are passed as arguments. This is an Ajax Event (/Ajax_Events/). Returning `false` in the `beforeSend` function will cancel the request. **As of jQuery 1.5**, the `beforeSend` option will be called regardless of the type of request.

---

cache (default: `true, false for dataType 'script' and 'jsonp'`)
Type: Boolean (http://api.jquery.com/Types/#Boolean)
If set to `false`, it will force requested pages not to be cached by the browser. **Note:** Setting `cache` to false will only work correctly with HEAD and GET requests. It works by appending "_={timestamp}" to the GET parameters. The parameter is not needed for other types of requests, except in IE8 when a POST is made to a URL that has already been requested by a GET.

---

complete
Type: Function (http://api.jquery.com/Types/#Function)(jqXHR (http://api.jquery.com/Types/#jqXHR) jqXHR, String (http://api.jquery.com/Types/#String) textStatus)
A function to be called when the request finishes (after `success` and `error` callbacks are executed). The function gets passed two arguments: The jqXHR (in jQuery 1.4.x, XMLHTTPRequest) object and a string categorizing the status of the request (`"success"`, `"notmodified"`, `"nocontent"`, `"error"`, `"timeout"`, `"abort"`, or `"parsererror"`). **As of jQuery 1.5**, the `complete` setting can accept an array of functions. Each function will be called in turn. This is an Ajax Event (/Ajax_Events/).

---

contents
Type: PlainObject (http://api.jquery.com/Types/#PlainObject)
An object of string/regular-expression pairs that determine how jQuery will parse the response, given its content type. **(version added: 1.5 (/category/version/1.5/))**

---

contentType (default: `'application/x-www-form-urlencoded; charset=UTF-8'`)
Type: Boolean (http://api.jquery.com/Types/#Boolean) or String (http://api.jquery.com/Types/#String)
When sending data to the server, use this content type. Default is "application/x-www-form-urlencoded; charset=UTF-8", which is fine for most cases. If you explicitly pass in a content-type to `$.ajax()`, then it is always sent to the server (even if no data is sent). As of jQuery 1.6 you can pass `false` to tell jQuery to not set any content type header. **Note:** The W3C XMLHttpRequest specification dictates that the charset is always UTF-8; specifying another charset will not force the browser to change the encoding. **Note:** For cross-domain requests, setting the content type to anything other than `application/x-www-form-urlencoded`, `multipart/form-data`, or `text/plain` will trigger the browser to send a preflight OPTIONS request to the server.

---

context
Type: PlainObject (http://api.jquery.com/Types/#PlainObject)
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" );
});
```

---

converters (default: `{"* text": window.String, "text html": true, "text json": jQuery.parseJSON, "text xml": jQuery.parseXML}`)
Type: PlainObject (http://api.jquery.com/Types/#PlainObject)

An object containing dataType-to-dataType converters. Each converter's value is a function that returns the transformed value of the response. **(version added: 1.5 (/category/version/1.5/))**

---

crossDomain (default: `false for same-domain requests, true for cross-domain requests`)
Type: Boolean (http://api.jquery.com/Types/#Boolean)
If you wish to force a crossDomain request (such as JSONP) on the same domain, set the value of crossDomain to `true`. This allows, for example, server-side redirection to another domain. **(version added: 1.5 (/category/version/1.5/))**

---

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

---

dataFilter
Type: Function (http://api.jquery.com/Types/#Function)(String (http://api.jquery.com/Types/#String) data, String (http://api.jquery.com/Types/#String) type) => Anything (http://api.jquery.com/Types/#Anything)
A function to be used to handle the raw response data of XMLHttpRequest. This is a pre-filtering function to sanitize the response. You should return the sanitized data. The function accepts two arguments: The raw data returned from the server and the 'dataType' parameter.

---

dataType (default: `Intelligent Guess (xml, json, script, or html)`)
Type: String (http://api.jquery.com/Types/#String)
The type of data that you're expecting back from the server. If none is specified, jQuery will try to infer it based on the MIME type of the response (an XML MIME type will yield XML, in 1.4 JSON will yield a JavaScript object, in 1.4 script will execute the script, and anything else will be returned as a string). The available types (and the result passed as the first argument to your success callback) are:

`"xml"`: Returns a XML document that can be processed via jQuery.

`"html"`: Returns HTML as plain text; included script tags are evaluated when inserted in the DOM.

`"script"`: Evaluates the response as JavaScript and returns it as plain text. Disables caching by appending a query string parameter, `_=[TIMESTAMP]`, to the URL unless the `cache` option is set to `true`. **Note:** This will turn POSTs into GETs for remote-domain requests.

`"json"`: Evaluates the response as JSON and returns a JavaScript object. Cross-domain `"json"` requests are converted to `"jsonp"` unless the request includes `jsonp: false` in its request options. The JSON data is parsed in a strict manner; any malformed JSON is rejected and a parse error is thrown. As of jQuery 1.9, an empty response is also rejected; the server should return a response of `null` or `{}` instead. (See json.org (http://json.org/) for more information on proper JSON formatting.)

`"jsonp"`: Loads in a JSON block using JSONP (http://bob.ippoli.to/archives/2005/12/05/remote-json-jsonp/). Adds an extra `"?callback=?"` to the end of your URL to specify the callback. Disables caching by appending a query string parameter, `"_=[TIMESTAMP]"`, to the URL unless the `cache` option is set to `true`.

`"text"`: A plain text string.

---

multiple, space-separated values: **As of jQuery 1.5**, jQuery can convert a dataType from what it received in the Content-Type header to what you require. For example, if you want a text response to be treated as XML, use `"text xml"` for the dataType. You can also make a JSONP request, have it received as text, and interpreted by jQuery as XML: `"jsonp text xml"`. Similarly, a shorthand string such as `"jsonp xml"` will first attempt to convert from jsonp to xml, and, failing that, convert from jsonp to text, and then from text to xml.

---

error
Type: Function (http://api.jquery.com/Types/#Function)(jqXHR (http://api.jquery.com/Types/#jqXHR) jqXHR, String (http://api.jquery.com/Types/#String) textStatus, String (http://api.jquery.com/Types/#String) errorThrown)

A function to be called if the request fails. The function receives three arguments: The jqXHR (in jQuery 1.4.x, XMLHttpRequest) object, a string describing the type of error that occurred and an optional exception object, if one occurred. Possible values for the second argument (besides `null`) are `"timeout"`, `"error"`, `"abort"`, and `"parsererror"`. When an HTTP error occurs, `errorThrown` receives the textual portion of the HTTP status, such as "Not Found" or "Internal Server Error." **As of jQuery 1.5**, the `error` setting can accept an array of functions. Each function will be called in turn. **Note:** *This handler is not called for cross-domain script and cross-domain JSONP requests.* This is an Ajax Event (/Ajax_Events/).

---

global (default: `true`)
Type: Boolean (http://api.jquery.com/Types/#Boolean)
Whether to trigger global Ajax event handlers for this request. The default is `true`. Set to `false` to prevent the global handlers like `ajaxStart` or `ajaxStop` from being triggered. This can be used to control various Ajax Events (/Ajax_Events/).

---

headers (default: `{}`)
Type: PlainObject (http://api.jquery.com/Types/#PlainObject)
An object of additional header key/value pairs to send along with requests using the XMLHttpRequest transport. The header `X-Requested-With: XMLHttpRequest` is always added, but its default `XMLHttpRequest` value can be changed here. Values in the `headers` setting can also be overwritten from within the `beforeSend` function. **(version added: 1.5 (/category/version/1.5/))**

---

ifModified (default: `false`)
Type: Boolean (http://api.jquery.com/Types/#Boolean)
Allow the request to be successful only if the response has changed since the last request. This is done by checking the Last-Modified header. Default value is `false`, ignoring the header. In jQuery 1.4 this technique also checks the 'etag' specified by the server to catch unmodified data.

---

isLocal (default: `depends on current location protocol`)
Type: Boolean (http://api.jquery.com/Types/#Boolean)
Allow the current environment to be recognized as "local," (e.g. the filesystem), even if jQuery does not recognize it as such by default. The following protocols are currently recognized as local: `file`, `*-extension`, and `widget`. If the `isLocal` setting needs modification, it is recommended to do so once in the `$.ajaxSetup()` method. **(version added: 1.5.1 (/category/version/1.5.1/))**

---

jsonp
Type: String (http://api.jquery.com/Types/#String) or Boolean (http://api.jquery.com/Types/#Boolean)
Override the callback function name in a JSONP request. This value will be used instead of 'callback' in the 'callback=?' part of the query string in the url. So `{jsonp:'onJSONPLoad'}` would result in `'onJSONPLoad=?'` passed to the server. **As of jQuery 1.5**, setting the `jsonp` option to `false` prevents jQuery from adding the "?callback" string to the URL or attempting to use "=?" for transformation. In this case, you should also explicitly set the `jsonpCallback` setting. For example, `{ jsonp: false, jsonpCallback: "callbackName" }`. If you don't trust the target of your Ajax requests, consider setting the `jsonp` property to `false` for security reasons.

---

jsonpCallback
Type: String (http://api.jquery.com/Types/#String) or Function (http://api.jquery.com/Types/#Function)()
Specify the callback function name for a JSONP request. This value will be used instead of the random name automatically generated by jQuery. It is preferable to let jQuery generate a unique name as it'll make it easier to manage the requests and provide callbacks and error handling. You may want to specify the callback when you want to enable better browser caching of GET requests. **As of jQuery 1.5**, you can also use a function for this setting, in which case the value of `jsonpCallback` is set to the return value of that function.

---

method (default: `'GET'`)
Type: String (http://api.jquery.com/Types/#String)
The HTTP method to use for the request (e.g. `"POST"`, `"GET"`, `"PUT"`). **(version added: 1.9.0 (/category/version/1.9.0/))**

---

mimeType
Type: String (http://api.jquery.com/Types/#String)
A mime type to override the XHR (XMLHttpRequest) mime type. **(version added: 1.5.1 (/category/version/1.5.1/))**

---

password
Type: String (http://api.jquery.com/Types/#String)
A password to be used with XMLHttpRequest in response to an HTTP access authentication request.

Version 1.5.1 (//api.jquery.com/category/version/1.5.1/)
Version 1.6 (//api.jquery.com/category/version/1.6/)
Version 1.7 (//api.jquery.com/category/version/1.7/)
Version 1.8 (//api.jquery.com/category/version/1.8/)
Version 1.9 (//api.jquery.com/category/version/1.9/)
Version 1.12 & 2.2 (//api.jquery.com/category/version/1.12-2.2/)
Version 3.0 (//api.jquery.com/category/version/3.0/)
Version 3.1 (//api.jquery.com/category/version/3.1/)

processData (default: `true`)
Type: Boolean (http://api.jquery.com/Types/#Boolean)
By default, data passed in to the `data` option as an object (technically, anything other than a string) will be processed and transformed into a query string, fitting to the default content-type "application/x-www-form-urlencoded". If you want to send a DOMDocument, or other non-processed data, set this option to `false`.

scriptCharset
Type: String (http://api.jquery.com/Types/#String)
Only applies when the "script" transport is used (e.g., cross-domain requests with "jsonp" or "script" dataType and "GET" type). Sets the `charset` attribute on the script tag used in the request. Used when the character set on the local page is not the same as the one on the remote script.

statusCode (default: `{}`)
Type: PlainObject (http://api.jquery.com/Types/#PlainObject)
An object of numeric HTTP codes and functions to be called when the response has the corresponding code. For example, the following will alert when the response status is a 404:

```
$.ajax({
  statusCode: {
    404: function() {
      alert( "page not found" );
    }
  }
});
```

If the request is successful, the status code functions take the same parameters as the success callback; if it results in an error (including 3xx redirect), they take the same parameters as the `error` callback.

(version added: 1.5 (/category/version/1.5/))

success
Type: Function (http://api.jquery.com/Types/#Function)(Anything (http://api.jquery.com/Types/#Anything) data, String (http://api.jquery.com/Types/#String) textStatus, jqXHR (http://api.jquery.com/Types/#jqXHR) jqXHR)
A function to be called if the request succeeds. The function gets passed three arguments: The data returned from the server, formatted according to the `dataType` parameter or the `dataFilter` callback function, if specified; a string describing the status; and the `jqXHR` (in jQuery 1.4.x, XMLHttpRequest) object. **As of jQuery 1.5**, *the success setting can accept an array of functions. Each function will be called in turn.* This is an Ajax Event (/Ajax_Events/).

timeout
Type: Number (http://api.jquery.com/Types/#Number)
Set a timeout (in milliseconds) for the request. A value of 0 means there will be no timeout. This will override any global timeout set with $.ajaxSetup() (/jQuery.ajaxSetup/). The timeout period starts at the point the `$.ajax` call is made; if several other requests are in progress and the browser has no connections available, it is possible for a request to time out before it can be sent. **In jQuery 1.4.x and below,** the XMLHttpRequest object will be in an invalid state if the request times out; accessing any object members may throw an exception. **In Firefox 3.0+ only,** script and JSONP requests cannot be cancelled by a timeout; the script will run even if it arrives after the timeout period.

traditional
Type: Boolean (http://api.jquery.com/Types/#Boolean)
Set this to `true` if you wish to use the traditional style of param serialization (/jQuery.param/).

type (default: `'GET'`)
Type: String (http://api.jquery.com/Types/#String)
An alias for `method`. You should use `type` if you're using versions of jQuery prior to 1.9.0.

url (default: `The current page`)
Type: String (http://api.jquery.com/Types/#String)
A string containing the URL to which the request is sent.

username
Type: String (http://api.jquery.com/Types/#String)
A username to be used with XMLHttpRequest in response to an HTTP access authentication request.

xhr (default: `ActiveXObject when available (IE), the XMLHttpRequest otherwise`)

Type: Function (http://api.jquery.com/Types/#Function)()
Callback for creating the XMLHttpRequest object. Defaults to the ActiveXObject when available (IE), the XMLHttpRequest otherwise. Override to provide your own implementation for XMLHttpRequest or enhancements to the factory.

---

xhrFields
Type: PlainObject (http://api.jquery.com/Types/#PlainObject)
An object of fieldName-fieldValue pairs to set on the native `XHR (XMLHttpRequest)` object. For example, you can use it to set `withCredentials` to `true` for cross-domain requests if needed.

```
$.ajax({
   url: a_cross_domain_url,
   xhrFields: {
      withCredentials: true
   }
});
```

In jQuery 1.5, the `withCredentials` property was not propagated to the native `XHR` and thus CORS requests requiring it would ignore this flag. For this reason, we recommend using jQuery 1.5.1+ should you require the use of it.

(version added: 1.5.1 (/category/version/1.5.1/))

The `$.ajax()` function underlies all Ajax requests sent by jQuery. It is often unnecessary to directly call this function, as several higher-level alternatives like `$.get() (/jQuery.get/)` and `.load() (/load/)` are available and are easier to use. If less common options are required, though, `$.ajax()` can be used more flexibly.

At its simplest, the `$.ajax()` function can be called with no arguments:

```
$.ajax();
```

Note: Default settings can be set globally by using the `$.ajaxSetup() (/jQuery.ajaxSetup/)` function.

This example, using no options, loads the contents of the current page, but does nothing with the result. To use the result, you can implement one of the callback functions.

The jqXHR Object

The jQuery XMLHttpRequest (jqXHR) object returned by `$.ajax()` **as of jQuery 1.5** is a superset of the browser's native XMLHttpRequest object. For example, it contains `responseText` and `responseXML` properties, as well as a `getResponseHeader()` method. When the transport mechanism is something other than XMLHttpRequest (for example, a script tag for a JSONP request) the `jqXHR` object simulates native XHR functionality where possible.

As of jQuery 1.5.1, the `jqXHR` object also contains the `overrideMimeType()` method (it was available in jQuery 1.4.x, as well, but was temporarily removed in jQuery 1.5). The `.overrideMimeType()` method may be used in the `beforeSend()` callback function, for example, to modify the response content-type header:

```
$.ajax({
  url: "http://fiddle.jshell.net/favicon.png",
  beforeSend: function( xhr ) {
    xhr.overrideMimeType( "text/plain; charset=x-user-defined" );
  }
})
  .done(function( data ) {
    if ( console && console.log ) {
      console.log( "Sample of data:", data.slice( 0, 100 ) );
    }
  });
```

The jqXHR objects returned by `$.ajax()` as of jQuery 1.5 implement the Promise interface, giving them all the properties, methods, and behavior of a Promise (see Deferred object (/category/deferred-object/) for more information). These methods take one or more function arguments that are called when the `$.ajax()` request terminates. This allows you to assign multiple callbacks on a single request, and even to assign callbacks after the request may have completed. (If the request is already complete, the callback is fired immediately.) Available Promise methods of the jqXHR object include:

jqXHR.done(function(data, textStatus, jqXHR) {});
An alternative construct to the success callback option, refer to `deferred.done() (/deferred.done/)` for implementation details.

jqXHR.fail(function(jqXHR, textStatus, errorThrown) {});

An alternative construct to the error callback option, the `.fail()` method replaces the deprecated `.error()` method. Refer to `deferred.fail()` (/deferred.fail/) for implementation details.

jqXHR.always(function(data|jqXHR, textStatus, jqXHR|errorThrown) { }); (added in jQuery 1.6)
An alternative construct to the complete callback option, the `.always()` method replaces the deprecated `.complete()` method.

In response to a successful request, the function's arguments are the same as those of `.done()`: data, textStatus, and the jqXHR object. For failed requests the arguments are the same as those of `.fail()`: the jqXHR object, textStatus, and errorThrown. Refer to `deferred.always()` (/deferred.always/) for implementation details.

jqXHR.then(function(data, textStatus, jqXHR) {}, function(jqXHR, textStatus, errorThrown) {});
Incorporates the functionality of the `.done()` and `.fail()` methods, allowing (as of jQuery 1.8) the underlying Promise to be manipulated. Refer to `deferred.then()` (/deferred.then/) for implementation details.

Deprecation Notice: The `jqXHR.success()`, `jqXHR.error()`, and `jqXHR.complete()` callbacks are removed as of jQuery 3.0. You can use `jqXHR.done()`, `jqXHR.fail()`, and `jqXHR.always()` instead.

```
// Assign handlers immediately after making the request,
// and remember the jqXHR object for this request
var jqxhr = $.ajax( "example.php" )
  .done(function() {
    alert( "success" );
  })
  .fail(function() {
    alert( "error" );
  })
  .always(function() {
    alert( "complete" );
  });

// Perform other work here ...

// Set another completion function for the request above
jqxhr.always(function() {
  alert( "second complete" );
});
```

The `this` reference within all callbacks is the object in the `context` option passed to `$.ajax` in the settings; if `context` is not specified, `this` is a reference to the Ajax settings themselves.

For backward compatibility with `XMLHttpRequest`, a `jqXHR` object will expose the following properties and methods:

`readyState`

`responseXML` and/or `responseText` when the underlying request responded with xml and/or text, respectively

`status`

`statusText`

`abort( [ statusText ] )`

`getAllResponseHeaders()` as a string

`getResponseHeader( name )`

`overrideMimeType( mimeType )`

`setRequestHeader( name, value )` which departs from the standard by replacing the old value with the new one rather than concatenating the new value to the old one

`statusCode( callbacksByStatusCode )`

No `onreadystatechange` mechanism is provided, however, since `done`, `fail`, `always`, and `statusCode` cover all conceivable requirements.

Callback Function Queues

The `beforeSend`, `error`, `dataFilter`, `success` and `complete` options all accept callback functions that are invoked at the appropriate times.

As of jQuery 1.5, the `fail` and `done`, and, as of jQuery 1.6, `always` callback hooks are first-in, first-out managed queues, allowing for more than one callback for each hook. See Deferred object methods (/category/deferred-object/), which are implemented internally for these

`$.ajax()` callback hooks.

The callback hooks provided by `$.ajax()` are as follows:

1. `beforeSend` callback option is invoked; it receives the `jqXHR` object and the `settings` object as parameters.
2. `error` callback option is invoked, if the request fails. It receives the `jqXHR`, a string indicating the error type, and an exception object if applicable. Some built-in errors will provide a string as the exception object: "abort", "timeout", "No Transport".
3. `dataFilter` callback option is invoked immediately upon successful receipt of response data. It receives the returned data and the value of `dataType`, and must return the (possibly altered) data to pass on to `success`.
4. `success` callback option is invoked, if the request succeeds. It receives the returned data, a string containing the success code, and the `jqXHR` object.
5. **Promise callbacks —** `.done()`, `.fail()`, `.always()`, and `.then()` **—** are invoked, in the order they are registered.
6. `complete` callback option fires, when the request finishes, whether in failure or success. It receives the `jqXHR` object, as well as a string containing the success or error code.

Data Types

Different types of response to `$.ajax()` call are subjected to different kinds of pre-processing before being passed to the success handler. The type of pre-processing depends by default upon the Content-Type of the response, but can be set explicitly using the `dataType` option. If the `dataType` option is provided, the Content-Type header of the response will be disregarded.

The available data types are `text`, `html`, `xml`, `json`, `jsonp`, and `script`.

If `text` or `html` is specified, no pre-processing occurs. The data is simply passed on to the success handler, and made available through the `responseText` property of the `jqXHR` object.

If `xml` is specified, the response is parsed using `jQuery.parseXML` (/jQuery.parseXML/) before being passed, as an `XMLDocument` (http://api.jquery.com/Types/#XMLDocument), to the success handler. The XML document is made available through the `responseXML` property of the `jqXHR` object.

If `json` is specified, the response is parsed using `jQuery.parseJSON` (/jQuery.parseJSON/) before being passed, as an object, to the success handler. The parsed JSON object is made available through the `responseJSON` property of the `jqXHR` object.

If `script` is specified, `$.ajax()` will execute the JavaScript that is received from the server before passing it on to the success handler as a string.

If `jsonp` is specified, `$.ajax()` will automatically append a query string parameter of (by default) `callback=?` to the URL. The `jsonp` and `jsonpCallback` properties of the settings passed to `$.ajax()` can be used to specify, respectively, the name of the query string parameter and the name of the JSONP callback function. The server should return valid JavaScript that passes the JSON response into the callback function. `$.ajax()` will execute the returned JavaScript, calling the JSONP callback function, before passing the JSON object contained in the response to the `$.ajax()` success handler.

For more information on JSONP, see the original post detailing its use (http://bob.pythonmac.org/archives/2005/12/05/remote-json-jsonp/).

Sending Data to the Server

By default, Ajax requests are sent using the GET HTTP method. If the POST method is required, the method can be specified by setting a value for the `type` option. This option affects how the contents of the `data` option are sent to the server. POST data will always be transmitted to the server using UTF-8 charset, per the W3C XMLHTTPRequest standard.

The `data` option can contain either a query string of the form `key1=value1&key2=value2`, or an object of the form `{key1: 'value1', key2: 'value2'}`. If the latter form is used, the data is converted into a query string using `jQuery.param() (/jQuery.param/)` before it is sent. This processing can be circumvented by setting `processData` to `false`. The processing might be undesirable if you wish to send an XML object to the server; in this case, change the `contentType` option from `application/x-www-form-urlencoded` to a more appropriate MIME type.

Advanced Options

The `global` option prevents handlers registered using `.ajaxSend() (/ajaxSend/)`, `.ajaxError() (/ajaxError/)`, and similar methods from firing when this request would trigger them. This can be useful to, for example, suppress a loading indicator that was implemented

with `.ajaxSend() (/ajaxSend/)` if the requests are frequent and brief. With cross-domain script and JSONP requests, the global option is automatically set to `false`. See the descriptions of these methods below for more details.

If the server performs HTTP authentication before providing a response, the user name and password pair can be sent via the `username` and `password` options.

Ajax requests are time-limited, so errors can be caught and handled to provide a better user experience. Request timeouts are usually either left at their default or set as a global default using `$.ajaxSetup() (/jQuery.ajaxSetup/)` rather than being overridden for specific requests with the `timeout` option.

By default, requests are always issued, but the browser may serve results out of its cache. To disallow use of the cached results, set `cache` to `false`. To cause the request to report failure if the asset has not been modified since the last request, set `ifModified` to `true`.

The `scriptCharset` allows the character set to be explicitly specified for requests that use a `<script>` tag (that is, a type of `script` or `jsonp`). This is useful if the script and host page have differing character sets.

The first letter in Ajax stands for "asynchronous," meaning that the operation occurs in parallel and the order of completion is not guaranteed. The `async` option to `$.ajax()` defaults to `true`, indicating that code execution can continue after the request is made. Setting this option to `false` (and thus making the call no longer asynchronous) is strongly discouraged, as it can cause the browser to become unresponsive.

The `$.ajax()` function returns the `XMLHttpRequest` object that it creates. Normally jQuery handles the creation of this object internally, but a custom function for manufacturing one can be specified using the `xhr` option. The returned object can generally be discarded, but does provide a lower-level interface for observing and manipulating the request. In particular, calling `.abort()` on the object will halt the request before it completes.

Extending Ajax

As of jQuery 1.5, jQuery's Ajax implementation includes prefilters (/jQuery.ajaxPrefilter/), transports (/jQuery.ajaxTransport/), and converters that allow you to extend Ajax with a great deal of flexibility.

Using Converters

`$.ajax()` converters support mapping data types to other data types. If, however, you want to map a custom data type to a known type (e.g `json`), you must add a correspondence between the response Content-Type and the actual data type using the `contents` option:

```
$.ajaxSetup({
  contents: {
    mycustomtype: /mycustomtype/
  },
  converters: {
    "mycustomtype json": function( result ) {
      // Do stuff
      return newresult;
    }
  }
});
```

This extra object is necessary because the response Content-Types and data types never have a strict one-to-one correspondence (hence the regular expression).

To convert from a supported type (e.g `text, json`) to a custom data type and back again, use another pass-through converter:

```
$.ajaxSetup({
  contents: {
    mycustomtype: /mycustomtype/
  },
  converters: {
    "text mycustomtype": true,
    "mycustomtype json": function( result ) {
      // Do stuff
      return newresult;
    }
  }
});
```

The above now allows passing from `text` to `mycustomtype` and then `mycustomtype` to `json`.

Additional Notes:

Due to browser security restrictions, most "Ajax" requests are subject to the same origin policy (http://en.wikipedia.org/wiki/Same_origin_policy); the request can not successfully

retrieve data from a different domain, subdomain, port, or protocol.

Script and JSONP requests are not subject to the same origin policy restrictions.

Examples:

Save some data to the server and notify the user once it's complete.

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

Retrieve the latest version of an HTML page.

```
$.ajax({
  url: "test.html",
  cache: false
})
  .done(function( html ) {
    $( "#results" ).append( html );
  });
```

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

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

xmlRequest.done( handleResponse );
```

Send an id as data to the server, save some data to the server, and notify the user once it's complete. If the request fails, alert the user.

```
var menuId = $( "ul.nav" ).first().attr( "id" );
var request = $.ajax({
  url: "script.php",
  method: "POST",
  data: { id : menuId },
  dataType: "html"
});

request.done(function( msg ) {
  $( "#log" ).html( msg );
});

request.fail(function( jqXHR, textStatus ) {
  alert( "Request failed: " + textStatus );
});
```

Load and execute a JavaScript file.

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