Should have strong knowledge of Python (scipy, numpy, pandas, sklearn and etc.) or Scala.

**What is CORS?**

Robert Gibb

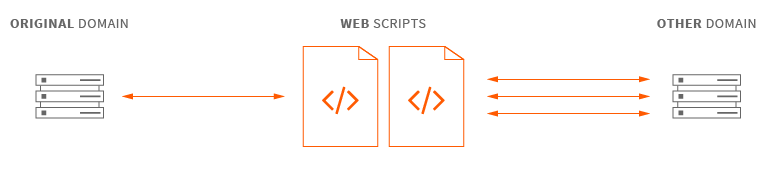
March 12, 2015

Updated

May 13, 2016

Definition

Cross-origin resource sharing (CORS) is a standard for accessing web resources on different domains. CORS allows web scripts to interact more openly with content outside of the original domain, leading to better integration between web services.



**Overview**

To prevent websites from tampering with each other, web browsers implement a security measure known as the same-origin policy. The same-origin policy lets resources (such as JavaScript) interact with resources from the same domain, but not with resources from a different domain. This provides security for the user by preventing abuse, such as running a script that reads the password field on a secure website.

In cases where cross-domain scripting is desired, CORS allows web developers to work around the same-origin policy. CORS adds HTTP headers which instruct web browsers on how to use and manage cross-domain content. The browser then allows or denies access to the content based on its security configuration.

**How CORS Works**

When a browser executes a script that references a resource on another domain, it requests the content directly from the second domain. The second domain determines whether or not to serve the content by validating the first domain, which is included as part of the request. The second domain then returns either the content or an error message back to the browser, bypassing the first domain entirely.

**Step-by-step, here’s how CORS works:**

1. A user opens a resource on a webpage which references another domain. This is usually a JavaScript file, but can include fonts and CSS resources.
2. The user’s browser creates a connection to the second domain, adding an “Origin” HTTP header to the request which contains the first domain.
3. The second domain replies with an “Access-Control-Allow-Origin” HTTP header which lists the domains allowed to make CORS requests. A wildcard (“\*”) allows all domains to make requests.
4. If the first domain is allowed to make the request, the second domain responds with the requested content.

The Access-Control-Allow-Origin header is defined in the second domain’s server configuration. If the header doesn’t contain wildcards and the first domain isn’t explicitly included, the browser displays an error message.

**Example of CORS**

CORS is an essential feature of online storage services such as Amazon S3. Service providers configure S3 to allow CORS requests from their website’s domain. When a user accesses the website and runs the script, their browser makes a request to S3. Since S3 is configured to allow the domain, the request is completed and the content is delivered to the browser.

More complicated cross-domain requests use “preflighting” to validate a request before the request is actually performed. Preflighted requests include custom HTTP headers which help the server determine whether or not a request is valid. Preflighted requests help service providers better protect sensitive information from users who might abuse CORS requests.

**Benefits of CORS**

CORS makes it easier for service providers to distribute content to users while adding interoperability to online services.

* **Users and enterprises experience greater interoperability between services,** allowing otherwise independent web services to cooperate and share resources.
* **Enterprises experience easier and more efficient resource management,** distributing web content across services in different domains.

**Conclusion**

Hackers are always looking for ways to undermine the security of the web. As of 2007, cross-site scripting, one of the most popular ways of bypassing the same origin policy, accounted for [84% of all security vulnerabilities](http://eval.symantec.com/mktginfo/enterprise/white_papers/b-whitepaper_exec_summary_internet_security_threat_report_xiii_04-2008.en-us.pdf) documented by Symantec. CORS gives service providers the freedom to interconnect web services without making their users vulnerable to attack.

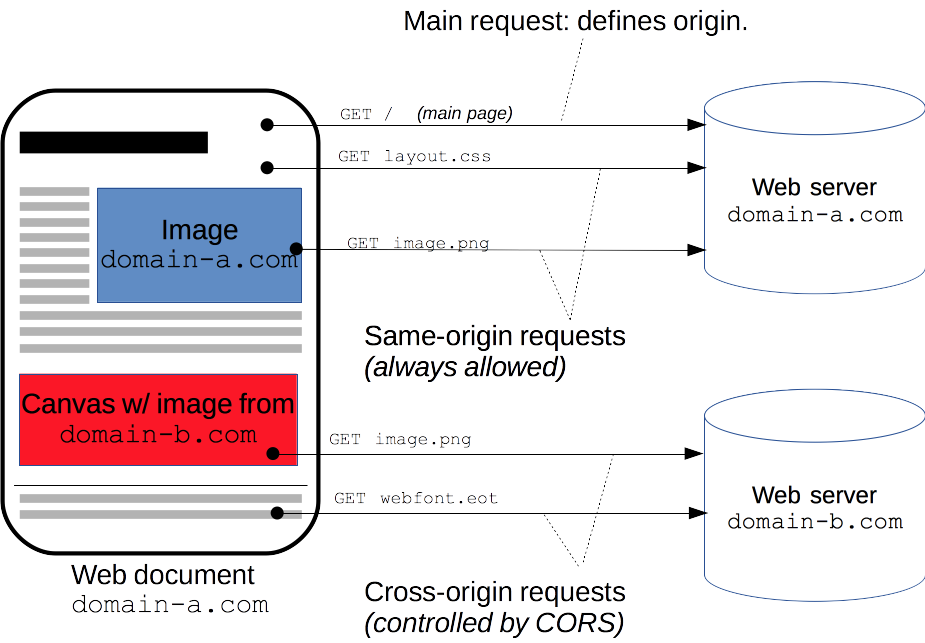
CORS makes it easy for web services to quickly and easily integrate without exposing their users.

Explain by mozila

Cross-Origin Resource Sharing ([CORS](https://developer.mozilla.org/en-US/docs/Glossary/CORS)) is a mechanism that uses additional [HTTP](https://developer.mozilla.org/en-US/docs/Glossary/HTTP) headers to tell a browser to let a web application running at one origin (domain) have permission to access selected resources from a server at a different origin. A web application makes a **cross-origin HTTP request** when it requests a resource that has a different origin (domain, protocol, and port) than its own origin.

An example of a cross-origin request: The frontend JavaScript code for a web application served from http://domain-a.com uses [XMLHttpRequest](https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest) to make a request for http://api.domain-b.com/data.json.

For security reasons, browsers restrict cross-origin HTTP requests initiated from within scripts. For example, [XMLHttpRequest](https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest) and the [Fetch API](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API) follow the [same-origin policy](https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy). This means that a web application using those APIs can only request HTTP resources from the same origin the application was loaded from, unless the response from the other origin includes the right CORS headers.



The CORS mechanism supports secure cross-origin requests and data transfers between browsers and web servers. Modern browsers use CORS in an API container such as [XMLHttpRequest](https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest) or [Fetch](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API) to help mitigate the risks of cross-origin HTTP requests.

## Who should read this article?

Everyone, really.

More specifically, this article is for web administrators, server developers, and front-end developers. Modern browsers handle the client-side components of cross-origin sharing, including headers and policy enforcement. But this new standard means servers have to handle new request and response headers. Another article for server developers discussing [cross-origin sharing from a server perspective (with PHP code snippets)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Server-Side_Access_Control) is supplementary reading.

## What requests use CORS?

This [cross-origin sharing standard](https://fetch.spec.whatwg.org/#http-cors-protocol) is used to enable cross-site HTTP requests for:

* Invocations of the [XMLHttpRequest](https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest) or [Fetch](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API) APIs in a cross-site manner, as discussed above.
* Web Fonts (for cross-domain font usage in @font-face within CSS), [so that servers can deploy TrueType fonts that can only be cross-site loaded and used by web sites that are permitted to do so.](https://www.w3.org/TR/css-fonts-3/#font-fetching-requirements)
* [WebGL textures](https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API/Tutorial/Using_textures_in_WebGL).
* Images/video frames drawn to a canvas using [drawImage](https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/drawImage).
* Stylesheets (for [CSSOM](https://developer.mozilla.org/en-US/docs/Web/CSS/CSSOM_View) access).
* Scripts (for unmuted exceptions).

This article is a general discussion of Cross-Origin Resource Sharing and includes a discussion of the necessary HTTP headers.

## Functional overview

The Cross-Origin Resource Sharing standard works by adding new [HTTP headers](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers) that allow servers to describe the set of origins that are permitted to read that information using a web browser.  Additionally, for HTTP request methods that can cause side-effects on server's data (in particular, for HTTP methods other than [GET](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/GET), or for [POST](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/POST) usage with certain [MIME types](https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types)), the specification mandates that browsers "preflight" the request, soliciting supported methods from the server with an HTTP [OPTIONS](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/OPTIONS) request method, and then, upon "approval" from the server, sending the actual request with the actual HTTP request method. Servers can also notify clients whether "credentials" (including [Cookies](https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies) and HTTP Authentication data) should be sent with requests.

Subsequent sections discuss scenarios, as well as provide a breakdown of the HTTP headers used.

## Examples of access control scenarios

Here, we present three scenarios that illustrate how Cross-Origin Resource Sharing works. All of these examples use the [XMLHttpRequest](https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest) object, which can be used to make cross-site invocations in any supporting browser.

The JavaScript snippets included in these sections (and running instances of the server-code that correctly handles these cross-site requests) can be found "in action" at <http://arunranga.com/examples/access-control/>, and will work in browsers that support cross-site [XMLHttpRequest](https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest).

A discussion of Cross-Origin Resource Sharing from a server perspective (including PHP code snippets) can be found in the [Server-Side Access Control (CORS)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Server-Side_Access_Control) article.

### Simple requests

Some requests don’t trigger a [CORS preflight](https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS#Preflighted_requests). Those are called “simple requests” in this article, though the [Fetch](https://fetch.spec.whatwg.org/) spec (which defines CORS) doesn’t use that term. A request that doesn’t trigger a [CORS preflight](https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS#Preflighted_requests)—a so-called “simple request”—is one that meets all the following conditions:

* The only allowed methods are:
  + [GET](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/GET)
  + [HEAD](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/HEAD)
  + [POST](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/POST)
* Apart from the headers set automatically by the user agent (for example, [Connection](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Connection), [User-Agent](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/User-Agent), or [any of the other headers with names defined in the Fetch spec as a “forbidden header name”](https://fetch.spec.whatwg.org/#forbidden-header-name)), the only headers which are allowed to be manually set are [those which the Fetch spec defines as being a “CORS-safelisted request-header”](https://fetch.spec.whatwg.org/#cors-safelisted-request-header), which are:
  + [Accept](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Accept)
  + [Accept-Language](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Accept-Language)
  + [Content-Language](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Language)
  + [Content-Type](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Type) (but note the additional requirements below)
  + [Last-Event-ID](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Last-Event-ID)
  + [DPR](http://httpwg.org/http-extensions/client-hints.html#dpr)
  + [Save-Data](http://httpwg.org/http-extensions/client-hints.html#save-data)
  + [Viewport-Width](http://httpwg.org/http-extensions/client-hints.html#viewport-width)
  + [Width](http://httpwg.org/http-extensions/client-hints.html#width)
* The only allowed values for the [Content-Type](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Type) header are:
  + application/x-www-form-urlencoded
  + multipart/form-data
  + text/plain
* No event listeners are registered on any [XMLHttpRequestUpload](https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequestUpload) object used in the request; these are accessed using the [XMLHttpRequest.upload](https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/upload) property.
* No [ReadableStream](https://developer.mozilla.org/en-US/docs/Web/API/ReadableStream) object is used in the request.

**Note:** These are the same kinds of cross-site requests that web content can already issue, and no response data is released to the requester unless the server sends an appropriate header. Therefore, sites that prevent cross-site request forgery have nothing new to fear from HTTP access control.

**Note:** WebKit Nightly and Safari Technology Preview place additional restrictions on the values allowed in the [Accept](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Accept), [Accept-Language](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Accept-Language), and [Content-Language](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Language) headers. If any of those headers have ”non-standard” values, WebKit/Safari does not consider the request to meet the conditions for a “simple request”. What WebKit/Safari considers “non-standard” values for those headers is not documented except in the following WebKit bugs: [Require preflight for non-standard CORS-safelisted request headers Accept, Accept-Language, and Content-Language](https://bugs.webkit.org/show_bug.cgi?id=165178), [Allow commas in Accept, Accept-Language, and Content-Language request headers for simple CORS](https://bugs.webkit.org/show_bug.cgi?id=165566), and [Switch to a blacklist model for restricted Accept headers in simple CORS requests](https://bugs.webkit.org/show_bug.cgi?id=166363). No other browsers implement those extra restrictions, because they’re not part of the spec.

For example, suppose web content on domain http://foo.example wishes to invoke content on domain http://bar.other. Code of this sort might be used within JavaScript deployed on foo.example:

var invocation = new XMLHttpRequest();

var url = 'http://bar.other/resources/public-data/';

function callOtherDomain() {

if(invocation) {

invocation.open('GET', url, true);

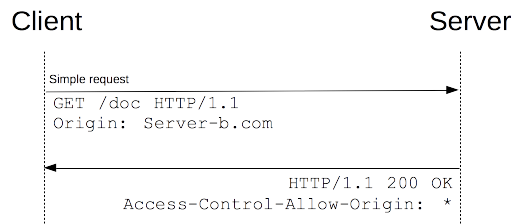
invocation.onreadystatechange = handler;

invocation.send();

}

}

This will lead to a simple exchange between the client and the server, using CORS headers to handle the privileges:



Let us look at what the browser will send to the server in this case, and let's see how the server responds:

GET /resources/public-data/ HTTP/1.1

Host: bar.other

User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.5; en-US; rv:1.9.1b3pre) Gecko/20081130 Minefield/3.1b3pre

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8

Accept-Language: en-us,en;q=0.5

Accept-Encoding: gzip,deflate

Accept-Charset: ISO-8859-1,utf-8;q=0.7,\*;q=0.7

Connection: keep-alive

Referer: http://foo.example/examples/access-control/simpleXSInvocation.html

Origin: http://foo.example

HTTP/1.1 200 OK

Date: Mon, 01 Dec 2008 00:23:53 GMT

Server: Apache/2.0.61

Access-Control-Allow-Origin: \*

Keep-Alive: timeout=2, max=100

Connection: Keep-Alive

Transfer-Encoding: chunked

Content-Type: application/xml

[XML Data]

Lines 1 - 10 are headers sent. The main HTTP request header of note here is the [Origin](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Origin) header on line 10 above, which shows that the invocation is coming from content on the domain http://foo.example.

Lines 13 - 22 show the HTTP response from the server on domain http://bar.other. In response, the server sends back an [Access-Control-Allow-Origin](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Access-Control-Allow-Origin) header, shown above in line 16. The use of the [Origin](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Origin) header and of [Access-Control-Allow-Origin](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Access-Control-Allow-Origin) show the access control protocol in its simplest use. In this case, the server responds with a Access-Control-Allow-Origin: \* which means that the resource can be accessed by **any** domain in a cross-site manner. If the resource owners at http://bar.other wished to restrict access to the resource to requests only from http://foo.example, they would send back:

Access-Control-Allow-Origin: http://foo.example

Note that now, no domain other than http://foo.example (identified by the ORIGIN: header in the request, as in line 10 above) can access the resource  in a cross-site manner.  The Access-Control-Allow-Origin header should contain the value that was sent in the request's Origin header.

### Preflighted requests

Unlike [“simple requests” (discussed above)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS#Simple_requests), "preflighted" requests first send an HTTP request by the [OPTIONS](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/OPTIONS) method to the resource on the other domain, in order to determine whether the actual request is safe to send. Cross-site requests are preflighted like this since they may have implications to user data.

In particular, a request is preflighted if **any of the following conditions** is true:

* **If** the request uses any of the following methods:
  + [PUT](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PUT)
  + [DELETE](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/DELETE)
  + [CONNECT](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/CONNECT)
  + [OPTIONS](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/OPTIONS)
  + [TRACE](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/TRACE)
  + [PATCH](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PATCH)
* **Or if**, apart from the headers set automatically by the user agent (for example, [Connection](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Connection), [User-Agent](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/User-Agent), or [any of the other header with a name defined in the Fetch spec as a “forbidden header name”](https://fetch.spec.whatwg.org/#forbidden-header-name)), the request includes any headers other than [those which the Fetch spec defines as being a “CORS-safelisted request-header”](https://fetch.spec.whatwg.org/#cors-safelisted-request-header), which are the following:
  + [Accept](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Accept)
  + [Accept-Language](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Accept-Language)
  + [Content-Language](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Language)
  + [Content-Type](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Type) (but note the additional requirements below)
  + [Last-Event-ID](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Last-Event-ID)
  + [DPR](http://httpwg.org/http-extensions/client-hints.html#dpr)
  + [Save-Data](http://httpwg.org/http-extensions/client-hints.html#save-data)
  + [Viewport-Width](http://httpwg.org/http-extensions/client-hints.html#viewport-width)
  + [Width](http://httpwg.org/http-extensions/client-hints.html#width)
* **Or if** the [Content-Type](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Type) header has a value other than the following:
  + application/x-www-form-urlencoded
  + multipart/form-data
  + text/plain
* **Or if** one or more event listeners are registered on an [XMLHttpRequestUpload](https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequestUpload) object used in the request.
* **Or if** a [ReadableStream](https://developer.mozilla.org/en-US/docs/Web/API/ReadableStream) object is used in the request.

**Note:** WebKit Nightly and Safari Technology Preview place additional restrictions on the values allowed in the [Accept](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Accept), [Accept-Language](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Accept-Language), and [Content-Language](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Language) headers. If any of those headers have ”non-standard” values, WebKit/Safari preflights the request. What WebKit/Safari considers “non-standard” values for those headers is not documented except in the following WebKit bugs: [Require preflight for non-standard CORS-safelisted request headers Accept, Accept-Language, and Content-Language](https://bugs.webkit.org/show_bug.cgi?id=165178), [Allow commas in Accept, Accept-Language, and Content-Language request headers for simple CORS](https://bugs.webkit.org/show_bug.cgi?id=165566), and [Switch to a blacklist model for restricted Accept headers in simple CORS requests](https://bugs.webkit.org/show_bug.cgi?id=166363). No other browsers implement those extra restrictions, because they’re not part of the spec.

The following is an example of a request that will be preflighted.

var invocation = new XMLHttpRequest();

var url = 'http://bar.other/resources/post-here/';

var body = '<?xml version="1.0"?><person><name>Arun</name></person>';

function callOtherDomain(){

if(invocation)

{

invocation.open('POST', url, true);

invocation.setRequestHeader('X-PINGOTHER', 'pingpong');

invocation.setRequestHeader('Content-Type', 'application/xml');

invocation.onreadystatechange = handler;

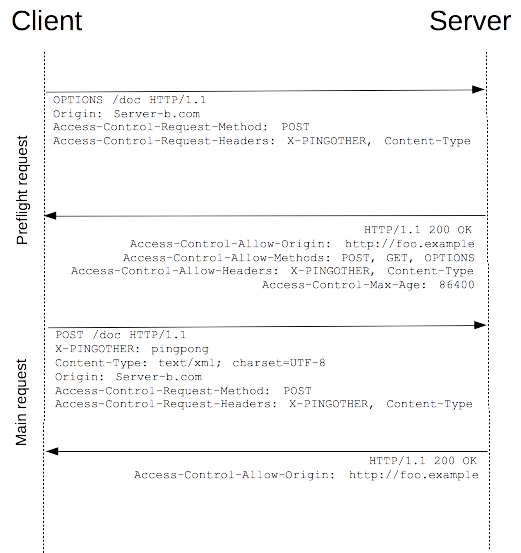
invocation.send(body);

}

}

......

In the example above, line 3 creates an XML body to send with the POST request in line 8.  Also, on line 9, a "customized" (non-standard) HTTP request header is set (X-PINGOTHER: pingpong).  Such headers are not part of the HTTP/1.1 protocol, but are generally useful to web applications.  Since the request uses a Content-Type of application/xml, and since a custom header is set, this request is preflighted.



(Note: as described below, the actual POST request does not include the Access-Control-Request-\* headers; they are needed only for the OPTIONS request.)

Let's take a look at the full exchange between client and server. The first exchange is the preflight request/response:

OPTIONS /resources/post-here/ HTTP/1.1

Host: bar.other

User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.5; en-US; rv:1.9.1b3pre) Gecko/20081130 Minefield/3.1b3pre

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8

Accept-Language: en-us,en;q=0.5

Accept-Encoding: gzip,deflate

Accept-Charset: ISO-8859-1,utf-8;q=0.7,\*;q=0.7

Connection: keep-alive

Origin: http://foo.example

Access-Control-Request-Method: POST

Access-Control-Request-Headers: X-PINGOTHER, Content-Type

HTTP/1.1 200 OK

Date: Mon, 01 Dec 2008 01:15:39 GMT

Server: Apache/2.0.61 (Unix)

Access-Control-Allow-Origin: http://foo.example

Access-Control-Allow-Methods: POST, GET

Access-Control-Allow-Headers: X-PINGOTHER, Content-Type

Access-Control-Max-Age: 86400

Vary: Accept-Encoding, Origin

Content-Encoding: gzip

Content-Length: 0

Keep-Alive: timeout=2, max=100

Connection: Keep-Alive

Content-Type: text/plain

Once the preflight request is complete, the real request is sent:

POST /resources/post-here/ HTTP/1.1

Host: bar.other

User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.5; en-US; rv:1.9.1b3pre) Gecko/20081130 Minefield/3.1b3pre

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8

Accept-Language: en-us,en;q=0.5

Accept-Encoding: gzip,deflate

Accept-Charset: ISO-8859-1,utf-8;q=0.7,\*;q=0.7

Connection: keep-alive

X-PINGOTHER: pingpong

Content-Type: text/xml; charset=UTF-8

Referer: http://foo.example/examples/preflightInvocation.html

Content-Length: 55

Origin: http://foo.example

Pragma: no-cache

Cache-Control: no-cache

<?xml version="1.0"?><person><name>Arun</name></person>

HTTP/1.1 200 OK

Date: Mon, 01 Dec 2008 01:15:40 GMT

Server: Apache/2.0.61 (Unix)

Access-Control-Allow-Origin: http://foo.example

Vary: Accept-Encoding, Origin

Content-Encoding: gzip

Content-Length: 235

Keep-Alive: timeout=2, max=99

Connection: Keep-Alive

Content-Type: text/plain

[Some GZIP'd payload]

Lines 1 - 12 above represent the preflight request with the [OPTIONS](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/OPTIONS) method. The browser determines that it needs to send this based on the request parameters that the JavaScript code snippet above was using, so that the server can respond whether it is acceptable to send the request with the actual request parameters. OPTIONS is an HTTP/1.1 method that is used to determine further information from servers, and is a [safe](https://developer.mozilla.org/en-US/docs/Glossary/safe) method, meaning that it can't be used to change the resource. Note that along with the OPTIONS request, two other request headers are sent (lines 10 and 11 respectively):

Access-Control-Request-Method: POST

Access-Control-Request-Headers: X-PINGOTHER, Content-Type

The [Access-Control-Request-Method](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Access-Control-Request-Method) header notifies the server as part of a preflight request that when the actual request is sent, it will be sent with a POST request method. The [Access-Control-Request-Headers](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Access-Control-Request-Headers) header notifies the server that when the actual request is sent, it will be sent with a X-PINGOTHER and Content-Type custom headers. The server now has an opportunity to determine whether it wishes to accept a request under these circumstances.

Lines 14 - 26 above are the response that the server sends back indicating that the request method (POST) and request headers (X-PINGOTHER) are acceptable. In particular, let's look at lines 17-20:

Access-Control-Allow-Origin: http://foo.example

Access-Control-Allow-Methods: POST, GET

Access-Control-Allow-Headers: X-PINGOTHER, Content-Type

Access-Control-Max-Age: 86400

The server responds with Access-Control-Allow-Methods and says that POST and GET are viable methods to query the resource in question. Note that this header is similar to the [Allow](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Allow) response header, but used strictly within the context of access control.

The server also sends Access-Control-Allow-Headers with a value of "X-PINGOTHER, Content-Type", confirming that these are permitted headers to be used with the actual request. Like Access-Control-Allow-Methods, Access-Control-Allow-Headers is a comma separated list of acceptable headers.

Finally, [Access-Control-Max-Age](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Access-Control-Max-Age) gives the value in seconds for how long the response to the preflight request can be cached for without sending another preflight request. In this case, 86400 seconds is 24 hours. Note that each browser has a [maximum internal value](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Access-Control-Max-Age) that takes precedence when the Access-Control-Max-Age is greater.

#### Preflighted requests and redirects

Most browsers currently don’t support following redirects for preflighted requests. If a redirect occurs for a preflighted request, most current browsers will report an error message such as the following.

The request was redirected to 'https://example.com/foo', which is disallowed for cross-origin requests that require preflight

Request requires preflight, which is disallowed to follow cross-origin redirect

The CORS protocol originally required that behavior but [was subsquently changed to no longer require it](https://github.com/whatwg/fetch/commit/0d9a4db8bc02251cc9e391543bb3c1322fb882f2). However, most browsers have not yet implemented the change and still exhibit the behavior that was originally required.

So until browsers catch up with the spec, you may be able to work around this limitation by doing one or both of the following:

* change the server-side behavior to avoid the preflight and/or to avoid the redirect—if you have control over the server the request is being made to
* change the request such that it is a [simple request](https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS#Simple_requests) that doesn’t cause a preflight

But if it’s not possible to make those changes, then another way that may be possible is to this:

1. Make a [simple request](https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS#Simple_requests) (using [Response.url](https://developer.mozilla.org/en-US/docs/Web/API/Response/url) for the Fetch API, or [XHR.responseURL](https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/responseURL)) to determine what URL the real preflighted request would end up at.
2. Make another request (the “real” request) using the URL you obtained from [Response.url](https://developer.mozilla.org/en-US/docs/Web/API/Response/url) or [XMLHttpRequest.responseURL](https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/responseURL) in the first step.

However, if the request is one that triggers a preflight due to the presence of the Authorization header in the request, you won’t be able to work around the limitation using the steps above. And you won’t be able to work around it at all unless you have control over the server the request is being made to.

### Requests with credentials

The most interesting capability exposed by both [XMLHttpRequest](https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest) or [Fetch](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API) and CORS is the ability to make "credentialed" requests that are aware of [HTTP cookies](https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies) and HTTP Authentication information. By default, in cross-site [XMLHttpRequest](https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest) or [Fetch](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API) invocations, browsers will **not** send credentials. A specific flag has to be set on the [XMLHttpRequest](https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest) object or the [Request](https://developer.mozilla.org/en-US/docs/Web/API/Request) constructor when it is invoked.

In this example, content originally loaded from http://foo.example makes a simple GET request to a resource on http://bar.other which sets Cookies. Content on foo.example might contain JavaScript like this:

var invocation = new XMLHttpRequest();

var url = 'http://bar.other/resources/credentialed-content/';

function callOtherDomain(){

if(invocation) {

invocation.open('GET', url, true);

invocation.withCredentials = true;

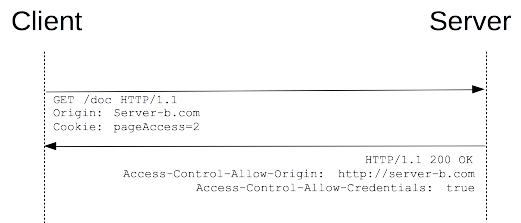
invocation.onreadystatechange = handler;

invocation.send();

}

}

Line 7 shows the flag on [XMLHttpRequest](https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest) that has to be set in order to make the invocation with Cookies, namely the withCredentials boolean value. By default, the invocation is made without Cookies. Since this is a simple GET request, it is not preflighted, but the browser will **reject** any response that does not have the [Access-Control-Allow-Credentials](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Access-Control-Allow-Credentials): true header, and **not** make the response available to the invoking web content.



Here is a sample exchange between client and server:

GET /resources/access-control-with-credentials/ HTTP/1.1

Host: bar.other

User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.5; en-US; rv:1.9.1b3pre) Gecko/20081130 Minefield/3.1b3pre

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8

Accept-Language: en-us,en;q=0.5

Accept-Encoding: gzip,deflate

Accept-Charset: ISO-8859-1,utf-8;q=0.7,\*;q=0.7

Connection: keep-alive

Referer: http://foo.example/examples/credential.html

Origin: http://foo.example

Cookie: pageAccess=2

HTTP/1.1 200 OK

Date: Mon, 01 Dec 2008 01:34:52 GMT

Server: Apache/2.0.61 (Unix) PHP/4.4.7 mod\_ssl/2.0.61 OpenSSL/0.9.7e mod\_fastcgi/2.4.2 DAV/2 SVN/1.4.2

X-Powered-By: PHP/5.2.6

Access-Control-Allow-Origin: http://foo.example

Access-Control-Allow-Credentials: true

Cache-Control: no-cache

Pragma: no-cache

Set-Cookie: pageAccess=3; expires=Wed, 31-Dec-2008 01:34:53 GMT

Vary: Accept-Encoding, Origin

Content-Encoding: gzip

Content-Length: 106

Keep-Alive: timeout=2, max=100

Connection: Keep-Alive

Content-Type: text/plain

[text/plain payload]

Although line 11 contains the Cookie destined for the content on http://bar.other, if bar.other did not respond with an [Access-Control-Allow-Credentials](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Access-Control-Allow-Credentials): true (line 19) the response would be ignored and not made available to web content.

#### Credentialed requests and wildcards

When responding to a credentialed request, the server **must** specify an origin in the value of the Access-Control-Allow-Origin header, instead of specifying the "\*" wildcard.

Because the request headers in the above example include a Cookie header, the request would fail if the value of the Access-Control-Allow-Origin header were "\*". But it does not fail: Because the value of the Access-Control-Allow-Origin header is "http://foo.example" (an actual origin) rather than the "\*" wildcard, the credential-cognizant content is returned to the invoking web content.

Note that the Set-Cookie response header in the example above also sets a further cookie. In case of failure, an exception—depending on the API used—is raised.

#### Third-party cookies

Note that cookies set in CORS responses are subject to normal third-party cookie policies. In the example above, the page is loaded from foo.example, but the cookie on line 22 is sent by bar.other, and would thus not be saved if the user has configured their browser to reject all third-party cookies.

## The HTTP response headers

This section lists the HTTP response headers that servers send back for access control requests as defined by the Cross-Origin Resource Sharing specification. The previous section gives an overview of these in action.

### Access-Control-Allow-Origin

A returned resource may have one [Access-Control-Allow-Origin](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Access-Control-Allow-Origin) header, with the following syntax:

Access-Control-Allow-Origin: <origin> | \*

The origin parameter specifies a URI that may access the resource. The browser must enforce this. For requests **without** credentials, the server may specify "\*" as a wildcard, thereby allowing any origin to access the resource.

For example, to allow http://mozilla.org to access the resource, you can specify:

Access-Control-Allow-Origin: http://mozilla.org

If the server specifies an origin host rather than "\*", then it could also include Origin in the Vary response header to indicate to clients that server responses will differ based on the value of the Origin request header.

### Access-Control-Expose-Headers

The [Access-Control-Expose-Headers](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Access-Control-Expose-Headers) header lets a server whitelist headers that browsers are allowed to access. For example:

Access-Control-Expose-Headers: X-My-Custom-Header, X-Another-Custom-Header

This allows the X-My-Custom-Header and X-Another-Custom-Header headers to be exposed to the browser.

### Access-Control-Max-Age

The  [Access-Control-Max-Age](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Access-Control-Max-Age" \o "The Access-Control-Max-Age response header indicates how long the results of a preflight request (that is the information contained in the Access-Control-Allow-Methods and Access-Control-Allow-Headers headers) can be cached.) header indicates how long the results of a preflight request can be cached. For an example of a preflight request, see the above examples.

Access-Control-Max-Age: <delta-seconds>

The delta-seconds parameter indicates the number of seconds the results can be cached.

### Access-Control-Allow-Credentials

The [Access-Control-Allow-Credentials](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Access-Control-Allow-Credentials) header Indicates whether or not the response to the request can be exposed when the credentials flag is true.  When used as part of a response to a preflight request, this indicates whether or not the actual request can be made using credentials. Note that simple GET requests are not preflighted, and so if a request is made for a resource with credentials, if this header is not returned with the resource, the response is ignored by the browser and not returned to web content.

Access-Control-Allow-Credentials: true

[Credentialed requests](https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS#Requests_with_credentials) are discussed above.

### Access-Control-Allow-Methods

The [Access-Control-Allow-Methods](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Access-Control-Allow-Methods) header specifies the method or methods allowed when accessing the resource. This is used in response to a preflight request. The conditions under which a request is preflighted are discussed above.

Access-Control-Allow-Methods: <method>[, <method>]\*

An example of a [preflight request is given above](https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS#Preflighted_requests), including an example which sends this header to the browser.

### Access-Control-Allow-Headers

The [Access-Control-Allow-Headers](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Access-Control-Allow-Headers) header is used in response to a [preflight request](https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS#Preflighted_requests) to indicate which HTTP headers can be used when making the actual request.

Access-Control-Allow-Headers: <field-name>[, <field-name>]\*

## The HTTP request headers

This section lists headers that clients may use when issuing HTTP requests in order to make use of the cross-origin sharing feature. Note that these headers are set for you when making invocations to servers. Developers using cross-site [XMLHttpRequest](https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest) capability do not have to set any cross-origin sharing request headers programmatically.

### Origin

The [Origin](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Origin) header indicates the origin of the cross-site access request or preflight request.

Origin: <origin>

The origin is a URI indicating the server from which the request initiated.  It does not include any path information, but only the server name.

**Note:** The origin can be the empty string; this is useful, for example, if the source is a data URL.

Note that in any access control request, the [Origin](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Origin) header is **always** sent.

### Access-Control-Request-Method

The [Access-Control-Request-Method](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Access-Control-Request-Method) is used when issuing a preflight request to let the server know what HTTP method will be used when the actual request is made.

Access-Control-Request-Method: <method>

Examples of this usage can be [found above.](https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS#Preflighted_requests)

### Access-Control-Request-Headers

The [Access-Control-Request-Headers](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Access-Control-Request-Headers) header is used when issuing a preflight request to let the server know what HTTP headers will be used when the actual request is made.

Access-Control-Request-Headers: <field-name>[, <field-name>]\*

Examples of this usage can be [found above](https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS#Preflighted_requests).