XMLHttpRequest

Read content offline
Did you know that you can read content offline by using one of these tools? If you would like to read offline MDN content in another format, let us know by commenting on Bug 665750.

`XMLHttpRequest` is a JavaScript object that was designed by Microsoft and adopted by Mozilla, Apple, and Google. It's now being standardized in the W3C. It provides an easy way to retrieve data from a URL without having to do a full page refresh. A Web page can update just a part of the 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, and it supports protocols other than HTTP (including `file` and `ftp`).

To create an instance of `XMLHttpRequest`, simply do this:

```
var myRequest = new XMLHttpRequest();
```

For details about how to use `XMLHttpRequest`, see Using XMLHttpRequest.

Method overview

```
XMLHttpRequest(JSObject objParameters);
void abort();
DOMString getAllResponseHeaders();
DOMString? getResponseHeader(DOMString header);
void open(DOMString method, DOMString url, optional boolean async,
optional DOMString? user, optional DOMString? password);
void overrideMimeType(DOMString mime);
void send();
void send(ArrayBuffer data);
void send(Blob data);
void send(Document data);
void send(DOMString? data);
void send(FormData data);
void setRequestHeader(DOMString header, DOMString value);
```

Non-standard methods

```
[noscript] void init(in nsIPrincipal principal, in nsIScriptContext
scriptContext, in nsPIDOMWindow ownerWindow);
```

```
[noscript] void openRequest(in AUTF8String method, in AUTF8String url, in
boolean async, in AString user, in AString password);
void sendAsBinary(in DOMString body);
```

Properties

Attribute	Type	Description
`onreadystatechange`	`Function?`	A JavaScript function object that is called whenever the `readyState` attribute changes. The callback is called from the user interface thread.
`readyState`	`unsigned short`	The state of the request:
		Value / State / Description
		`0` / `UNSENT` / `open()` has not been called yet.
		`1` / `OPENED` / `send()` has not been called yet.
		`2` / `HEADERS_RECEIVED` / `send()` has been called, and headers and status are available.
		`3` / `LOADING` / Downloading; `responseText` holds partial data.
		`4` / `DONE` / The operation is complete.
`response`	varies	The response entity body according to `responseType`, as an `ArrayBuffer`, `Blob`, `Document`, JavaScript object (for "json"), or string. This is `null` if the request is not complete or was not successful.
`responseText` Read only	`DOMString`	The response to the request as text, or `null` if the request was unsuccessful or has not yet been sent.
`responseType`	`XMLHttpRequestResponseType`	Can be set to change the response type.
		Value / Data type of `response` property
		`""` (empty string) / String (this is the default)
		`"arraybuffer"` / `ArrayBuffer`
		`"blob"` / `Blob`
		`"document"` / `Document`

`"json"`	JavaScript object, parsed from a JSON string returned by the server
`"text"`	String
`"moz-blob"`	Used by Firefox to allow retrieving partial `Blob` data from progress events. This lets your progress event handler start processing data while it's still being received.
`"moz-chunked-text"`	Similar to `"text"`, but is streaming. This means that the value in `response`, is only available during dispatch of the `"progress"` event and only contains the data received since the last `"progress"` event. When `response` is accessed during a `"progress"` event it contains a string with the data. Otherwise it returns `null`. This mode currently only works in Firefox.
`"moz-chunked-arraybuffer"`	Similar to `"arraybuffer"`, but is streaming. This means that the value in `response`, is only available during dispatch of the `"progress"` event and only contains the data received since the last

		`"progress"` event. When `response` is accessed during a `"progress"` event it contains a string with the data. Otherwise it returns `null`. This mode currently only works in Firefox. **Note:** Starting with Gecko 11.0 (Firefox 11.0 / Thunderbird 11.0 / SeaMonkey 2.8), as well as WebKit build 528, these browsers no longer let you use the `responseType` attribute when performing synchronous requests. Attempting to do so throws an `NS_ERROR_DOM_INVALID_ACCESS_ERR` exception. This change has been proposed to the W3C for standardization.
`responseXML` Read only	`Document?`	The response to the request as a DOM `Document` object, or `null` if the request was unsuccessful, has not yet been sent, or cannot be parsed as XML or HTML. The response is parsed as if it were a `text/xml` stream. When the `responseType` is set to `"document"` and the request has been made asynchronously, the response is parsed as it were a `text/html` stream. **Note:** If the server doesn't apply the `text/xml` Content-Type header, you can use `overrideMimeType()`to force `XMLHttpRequest` to parse it as XML anyway.
`status` Read only	`unsigned short`	The status of the response to the request. This is the HTTP result code (for example, `status` is 200 for a successful request).
`statusText` Read only	`DOMString`	The response string returned by the HTTP server. Unlike `status`, this includes the entire text of the response message ("`200 OK`", for example).
`timeout`	`unsigned long`	The number of milliseconds a request can take before automatically being terminated. A value of 0 (which is the default) means there is no timeout. **Note:** You may not use a timeout for synchronous requests with an owning

		window.
`upload`	`XMLHttpRequestUpload`	The upload process can be tracked by adding an event listener to `upload`.
`withCredentials`	`boolean`	Indicates whether or not cross-site `Access-Control` requests should be made using credentials such as cookies or authorization headers. The default is `false`. **Note:** This never affects same-site requests. **Note:** Starting with Gecko 11.0 (Firefox 11.0 / Thunderbird 11.0 / SeaMonkey 2.8), Gecko no longer lets you use the `withCredentials` attribute when performing synchronous requests. Attempting to do so throws an `NS_ERROR_DOM_INVALID_ACCESS_ERR` exception.

Non-standard properties

Attribute	Type	Description
`channel` Read only	`nsIChannel`	The channel used by the object when performing the request. This is `null` if the channel hasn't been created yet. In the case of a multi-part request, this is the initial channel, not the different parts in the multi-part request. **Requires elevated privileges to access.**
`mozAnon` Read only	`boolean`	If true, the request will be sent without cookie and authentication headers.
`mozSystem` Read only	`boolean`	If true, the same origin policy will not be enforced on the request.
`mozBackgroundRequest`	`boolean`	Indicates whether or not the object represents a background service request. If `true`, no load group is associated with the request, and security dialogs are prevented from being shown to the user. **Requires elevated privileges to access.** In cases in which a security dialog (such as authentication or a bad certificate notification) would normally be shown, the request simply fails instead. **Note:** This property must be set before calling `open()`.
`mozResponseArrayBuffer` Obsolete since Gecko 6 Read only	`ArrayBuffer`	The response to the request, as a JavaScript typed array. This is `NULL` if the request was not successful, or if it hasn't been sent yet.
`multipart` Obsolete since Gecko 22	`boolean`	**This Gecko-only feature was removed in Firefox/Gecko 22.** Please use Server-Sent Events, Web Sockets or `responseText` from progress events instead.

Indicates whether or not the response is expected to be a stream of possibly multiple XML documents. If set to `true`, the content type of the initial response must be `multipart/x-mixed-replace` or an error will occur. All requests must be asynchronous.

This enables support for server push; for each XML document that's written to this request, a new XML DOM document is created and the `onload` handler is called between documents.

Note: When this is set, the `onload` handler and other event handlers are not reset after the first XMLdocument is loaded, and the `onload` handler is called after each part of the response is received.

Constructor

XMLHttpRequest()

The constructor initiates a XMLHttpRequest. It must be called before any other method calls.

Gecko/Firefox 16 adds a non-standard parameter to the constructor that can enable anonymous mode (see Bug 692677). Setting the `mozAnon` flag to `true` effectively resembles the `AnonXMLHttpRequest()` constructor described in the XMLHttpRequest specification which has not been implemented in any browser yet (as of September 2012).

```
XMLHttpRequest (
  JSObject objParameters
);
```

Parameters (non-standard)

`objParameters` **Requires Gecko 16.0**

There are two flags you can set:

`mozAnon`

Boolean: Setting this flag to `true` will cause the browser not to expose the origin and user credentials when fetching resources. Most important, this means that cookies will not be sent unless explicitly added using setRequestHeader.

`mozSystem`

Boolean: Setting this flag to `true` allows making cross-site connections without requiring the server to opt-in using CORS. *Requires setting `mozAnon: true`. I.e. this can't be combined with sending cookies or other user credentials. This only works in privileged (reviewed) apps; it does not work on arbitrary webpages loaded in Firefox.*

Methods

abort()

Aborts the request if it has already been sent.

getAllResponseHeaders()

```
DOMString getAllResponseHeaders();
```

Returns all the response headers as a string, or `null` if no response has been received. **Note:** For multipart requests, this returns the headers from the *current* part of the request, not from the original channel.

getResponseHeader()

```
DOMString? getResponseHeader(DOMString header);
```

Returns the string containing the text of the specified header, or `null` if either the response has not yet been received or the header doesn't exist in the response.

open()

Initializes a request. This method is to be used from JavaScript code; to initialize a request from native code, use `openRequest()`instead.

Note: Calling this method an already active request (one for which `open()`or `openRequest()`has already been called) is the equivalent of calling `abort()`.

```
void open(
   DOMString method,
   DOMString url,
   optional boolean async,
   optional DOMString user,
   optional DOMString password
);
```

Parameters

`method`

The HTTP method to use, such as "GET", "POST", "PUT", "DELETE", etc. Ignored for non-HTTP(S) URLs.

`url`

The URL to which to send the request.

`async`

An optional boolean parameter, defaulting to `true`, indicating whether or not to perform the operation asynchronously. If this value is `false`, the `send()`method does not return until the response is received. If `true`, notification of a completed transaction is provided using event listeners. This *must* be true if the `multipart` attribute is `true`, or an exception will be thrown.

`user`

The optional user name to use for authentication purposes; by default, this is an empty string.

`password`

The optional password to use for authentication purposes; by default, this is an empty string.

overrideMimeType()

Overrides the MIME type returned by the server. This may be used, for example, to force a stream to be treated and parsed as text/xml, even if the server does not report it as such.This method must be called before `send()`.

```
void overrideMimeType(DOMString mimetype);
```

send()

Sends the request. If the request is asynchronous (which is the default), this method returns as soon as the request is sent. If the request is synchronous, this method doesn't return until the response has arrived.

Note: Any event listeners you wish to set must be set before calling `send()`.

```
void send();
void send(ArrayBuffer data);
void send(Blob data);
void send(Document data);
void send(DOMString? data);
void send(FormData data);
```

Notes

If the *data* is a `Document`, it is serialized before being sent. When sending a Document, versions of Firefox prior to version 3 always send the request using UTF-8 encoding; Firefox 3 properly sends the document using the encoding specified by `body.xmlEncoding`, or UTF-8 if no encoding is specified.

If it's an `nsIInputStream`, it must be compatible with `nsIUploadChannel`'s `setUploadStream()` method. In that case, a Content-Length header is added to the request, with its value obtained using `nsIInputStream`'s `available()` method. Any headers included at the top of the stream are treated as part of the message body. The stream's MIMEtype should be specified by setting the Content-Type header using the `setRequestHeader()` method prior to calling `send()`.

The best way to send binary content (like in files upload) is using ArrayBuffers or Blobs in conjuncton with the `send()` method. However, if you want to send a stringifiable raw data, use the `sendAsBinary()` method instead, or the `StringView` Non native typed arrays superclass.

setRequestHeader()

Sets the value of an HTTP request header.You must call `setRequestHeader()` after `open()`, but before `send()`. If this method is called several times with the same header, the values are merged into one single request header.

```
void setRequestHeader(
   DOMString header,
   DOMString value
);
```

Parameters

`header`

The name of the header whose value is to be set.

`value`

The value to set as the body of the header.

Non-standard methods

init()

Initializes the object for use from C++code.

```
[noscript] void init(
   in nsIPrincipal principal,
   in nsIScriptContext scriptContext,
   in nsPIDOMWindow ownerWindow
);
```

Parameters

`principal`

The principal to use for the request; must not be `null`.

`scriptContext`

The script context to use for the request; must not be `null`.

`ownerWindow`

The window associated with the request; may be `null`.

openRequest()

Initializes a request. This method is to be used from native code; to initialize a request from JavaScript code, use `open()`instead. See the documentation for `open()`.

sendAsBinary()

Requires Gecko 1.9 (Firefox 3)

A variant of the `send()` method that sends binary data.

```
void sendAsBinary(
   in DOMString body
);
```

This method, used in conjuncton with the `readAsBinaryString` method of the `FileReader` API make possible to read and **upload** any type of file and to stringify the raw data.

Parameters

`body`

The request body as a DOMstring. This data is converted to a string of single-byte characters by truncation (removing the high-order byte of each character).

sendAsBinary() polyfill

Since `sendAsBinary()` is an experimental feature, here is **a polyfill** for browsers which *don't* support the `sendAsBinary()` method but support typed arrays.

```
if (!XMLHttpRequest.prototype.sendAsBinary) {
  XMLHttpRequest.prototype.sendAsBinary = function (sData) {
    var nBytes = sData.length, ui8Data = new Uint8Array(nBytes);
    for (var nIdx = 0; nIdx < nBytes; nIdx++) {
      ui8Data[nIdx] = sData.charCodeAt(nIdx) & 0xff;
    }

    this.send(ui8Data);

  };
}
```

Note: It's possible to build this polyfill putting two types of data as argument for `send()`: an `ArrayBuffer` (`ui8Data.buffer` – the commented code) or an `ArrayBufferView` (`ui8Data`, which is a typed array of 8-bit unsigned integers – uncommented code). However, on Google Chrome, when you try to send an `ArrayBuffer`, the following warning message will appear: `ArrayBuffer is deprecated in XMLHttpRequest.send(). Use ArrayBufferView instead.` Another possible approach to send binary data is the `StringView` Non native typed arrays superclass in conjunction with the `send()` method.

Notes

- By default, Firefox 3 limits the number of `XMLHttpRequest` connections per server to 6 (previous versions limit this to 2 per server). Some interactive web sites may keep an `XMLHttpRequest` connection open, so opening multiple sessions to such sites may result in the browser hanging in such a way that the window no longer repaints and controls don't respond. This value can be changed by editing the `network.http.max-persistent-connections-per-server` preference in `about:config`.
- From Gecko 7.0 headers set by `setRequestHeader()` are sent with the request when following a redirect. Previously these headers would not be sent.
- `XMLHttpRequest` is implemented in Gecko using the `nsIXMLHttpRequest`, `nsIXMLHttpRequestEventTarget`, and `nsIJSXMLHttpRequest` interfaces.

Events

`onreadystatechange` as a property of the `XMLHttpRequest` instance is supported in all browsers.

Since then, a number of additional event handlers were implemented in various browsers (`onload`, `onerror`, `onprogress`, etc.). These are supported in Firefox. In particular, see `nsIXMLHttpRequestEventTarget` and Using XMLHttpRequest.

More recent browsers, including Firefox, also support listening to the `XMLHttpRequest` events via standard `addEventListener` APIs in addition to setting `on*` properties to a handler function.

Browser compatibility

Desktop

Feature	Chrome	Firefox (Gecko)	Internet Explorer	Opera	Safari (WebKit)
Basic support (XHR1)	1	1.0	5 (via ActiveXObject) 7 (XMLHttpRequest)	(Yes)	1.2
`send(ArrayBuffer)`	9	9	10	11.60	?
`send(Blob)`	7	3.6	10	12	?
`send(FormData)`	6	4	10	12	?
`sendAsBinary(DOMString)`	Not supported – use the polyfill	1.9	Not supported	Not supported	Not supported
`response`	10	6	10	11.60	?
`responseType` = 'arraybuffer'	10	6	10	11.60	?
`responseType` = 'blob'	19	6	10	12	?
`responseType` = 'document'	18	11	10	Not supported	Not supported
`responseType` = 'json'	Not supported	10	Not supported	12	Not supported
Progress Events	7	3.5	10	12	?
`withCredentials`	3	3.5	10	12	4
`timeout`	29	12.0	8	12	Not supported
`responseType` = 'moz-blob'	Not supported	12.0	Not supported	Not supported	Not supported

Gecko notes

Gecko 11.0 (Firefox 11.0 / Thunderbird 11.0 / SeaMonkey 2.8) removed support for using the `responseType` and `withCredentials` attributes when performing synchronous requests. Attempting to do so throws an `NS_ERROR_DOM_INVALID_ACCESS_ERR` exception. This change has been proposed to the W3C for standardization.

Gecko 12.0 (Firefox 12.0 / Thunderbird 12.0 / SeaMonkey 2.9) and later support using `XMLHttpRequest` to read from `data:` URLs.

See also

- MDN articles about XMLHttpRequest:
  - AJAX - Getting Started
  - Using XMLHttpRequest
  - HTML in XMLHttpRequest

  - `FormData`

- XMLHttpRequest references from W3C and browser vendors:
  - W3C: XMLHttpRequest (base features)
  - W3C: XMLHttpRequest (latest editor's draft with extensions to the base functionality, formerly XMLHttpRequest Level 2
  - Microsoft documentation
  - Apple developers' reference

- "Using the XMLHttpRequest Object" (jibbering.com)
- XMLHttpRequest - REST and the Rich User Experience
- HTML5 Rocks - New Tricks in XMLHttpRequest2

d **Tags (5)**

- AJAX
- XMLHttpRequest
- NeedsMobileBrowserCompatibility
- MakeBrowserAgnostic
- HTTP

Contributors to this page: mwahaha, myakura, Dbs, fusionchess, Jürgen Jeka, MarkFinkle, alispivak, Np, Yaffle, Chbok, emk, kmaglione, Mak77, another_sam, guirong, Okome, khuey, julienfabre, saneyuki_s, kathyw, Inimino, jkronegg, Riboribo, XP1, Sicking, Sevenspade, markrgrant, Brettz9, didizingo, Konno.Software, sdwilsh, Cfinke, psdmca, Ted_Mielczarek, LukSoft, Cipherzero, Vesicles, Carrie zhxj, Prahal, Alaa.moustafa, B0at, Bruceboughton, LeeHyeok, Lev.novikov, Sheppy, tymofiy, natevw, ericjung, fbender, mattbasta, ziyunfei, geoffstokes, Nickolay, trevorh, Simon Lindholm, Mossop, MarkGiffin, ebidel, Waldo, teoli, matt.kantor, darktrojan, grendel, Hsivonen, MatrixFrog, CN, Gandalf, Asuza, Federico, Mgjbot, TwelveBaud, Dria, timdream, Bzbarsky
Last updated by: teoli, Sep 14, 2013 4:32:15 PM
Last reviewed by: teoli, Sep 14, 2013 4:32:15 PM