Fetch

Living Standard — Last Updated 23 March 2019

Participate:

GitHub whatwg/fetch (new issue, open issues)
IRC: #whatwg on Freenode

Commits:

GitHub whatwg/fetch/commits
Snapshot as of this commit
@fetchstandard

Tests:

web-platform-tests fetch/ (ongoing work)

Translations (non-normative)**:**

日本語

File an issue about the selected text

Abstract

The Fetch standard defines requests, responses, and the process that binds them: fetching.

Table of Contents

File an issue about the selected text

Goals

To unify fetching across the web platform this specification supplants a number of algorithms and specifications:

- HTML Standard’s fetch and potentially CORS-enabled fetch algorithms [HTML]
- CORS [CORS]
- HTTP `Origin` header semantics [ORIGIN]

Unifying fetching provides consistent handling of:

- URL schemes
- Redirects
- Cross-origin semantics
- CSP [CSP]
- Service workers [SW]
- Mixed Content [MIX]
- `Referer` [REFERRER]

1. Preface §

At a high level, fetching a resource is a fairly simple operation. A request goes in, a response comes out. The details of that operation are however quite involved and used to not be written down carefully and differ from one API to the next.

Numerous APIs provide the ability to fetch a resource, e.g. HTML’s `img` and `script` element, CSS' `cursor` and `list-style-image`, the `navigator.sendBeacon()` and `self.importScripts()` JavaScript APIs. The Fetch Standard provides a unified architecture for these features so they are all consistent when it comes to various aspects of fetching, such as redirects and the CORS protocol.

The Fetch Standard also defines the `fetch()` JavaScript API, which exposes most of the networking functionality at a fairly low level of abstraction.

File an issue about the selected text

2. Infrastructure §

This specification depends on the Infra Standard. [INFRA]

This specification uses terminology from the ABNF, Encoding, HTML, HTTP, IDL, MIME Sniffing, Streams, and URL Standards. [ABNF] [ENCODING] [HTML] [HTTP] [WEBIDL] [MIMESNIFF] [STREAMS] [URL]

ABNF means ABNF as augmented by HTTP (in particular the addition #) and RFC 7405. [RFC7405]

Credentials are HTTP cookies, TLS client certificates, and authentication entries (for HTTP authentication). [COOKIES] [TLS] [HTTP-AUTH]

Tasks that are queued by this standard are annotated as one of:

- **process request body**
- **process request end-of-body**
- **process response**
- **process response end-of-body**
- **process response done**

To **queue a fetch task** on request *request* to *run an operation*, run these steps:

1. If *request*'s client is null, terminate these steps.
2. Queue a task to *run an operation* on *request*'s client's responsible event loop using the networking task source.

To **queue a fetch-request-done task**, given a *request*, queue a fetch task on *request* to process request end-of-body for *request*.

To **serialize an integer**, represent it as a string of the shortest possible decimal number.

This will be replaced by a more descriptive algorithm in Infra. See infra/201.

2.1. URL §

A **local scheme** is a scheme that is "`about`", "`blob`", or "`data`".

A URL **is local** if its scheme is a local scheme.

Note

This definition is also used by Referrer Policy. [REFERRER]

An **HTTP(S) scheme** is a scheme that is "`http`" or "`https`".

A **network scheme** is a scheme that is "`ftp`" or an HTTP(S) scheme.

A **fetch scheme** is a scheme that is "`about`", "`blob`", "`data`", "`file`", or a network scheme.

Note

HTTP(S) scheme, network scheme, and fetch scheme are also used by HTML. [HTML]

A **response URL** is a URL for which implementations need not store the fragment as it is never exposed. When serialized, the *exclude fragment flag* is set, meaning implementations can store the fragment nonetheless.

File an issue about the selected text

2.2. HTTP §

While fetching encompasses more than just HTTP, it borrows a number of concepts from HTTP and applies these to resources obtained via other means (e.g., `data` URLs).

An **HTTP tab or space** is U+0009 TAB or U+0020 SPACE.

HTTP whitespace is U+000A LF, U+000D CR, or an HTTP tab or space.

Note

HTTP whitespace is only useful for specific constructs that are reused outside the context of HTTP headers (e.g., MIME types). For HTTP header values, using HTTP tab or space is preferred, and outside that context ASCII whitespace is preferred. Unlike ASCII whitespace this excludes U+000C FF.

An **HTTP newline byte** is 0x0A (LF) or 0x0D (CR).

An **HTTP tab or space byte** is 0x09 (HT) or 0x20 (SP).

An **HTTP whitespace byte** is an HTTP newline byte or HTTP tab or space byte.

An **HTTPS state value** is "`none`", "`deprecated`", or "`modern`".

Note

A response delivered over HTTPS will typically have its HTTPS state set to "`modern`". A user agent can use "`deprecated`" in a transition period. E.g., while removing support for a hash function, weak cipher suites, certificates for an "Internal Name", or certificates with an overly long validity period. How exactly a user agent can use "`deprecated`" is not defined by this specification. An environment settings object typically derives its HTTPS state from a response.

To **collect an HTTP quoted string** from a string *input*, given a position variable *position* and optionally an *extract-value flag*, run these steps:

1. Let *positionStart* be *position*.
2. Let *value* be the empty string.
3. Assert: the code point at *position* within *input* is U+0022 (").
4. Advance *position* by 1.
5. While true:
    1. Append the result of collecting a sequence of code points that are not U+0022 (") or U+005C (\) from *input*, given *position*, to *value*.
    2. If *position* is past the end of *input*, then break.
    3. Let *quoteOrBackslash* be the code point at *position* within *input*.
    4. Advance *position* by 1.
    5. If *quoteOrBackslash* is U+005C (\), then:
        1. If *position* is past the end of *input*, then append U+005C (\) to *value* and break.
        2. Append the code point at *position* within *input* to *value*.
        3. Advance *position* by 1.
    6. Otherwise:
        1. Assert: *quoteOrBackslash* is U+0022 (").
        2. Break.
6. If the *extract-value flag* is set, then return *value*.
7. Return the code points from *positionStart* to *position*, inclusive, within *input*.

Note

The extract-value flag *argument makes this algorithm suitable for getting, decoding, and splitting and parse a MIME type, as well as other header value parsers that might need this.*

2.2.1. Methods §

File an issue about the selected text

A **method** is a byte sequence that matches the method token production.

A **CORS-safelisted method** is a method that is `` `GET` ``, `` `HEAD` ``, or `` `POST` ``.

A **forbidden method** is a method that is a byte-case-insensitive match for `` `CONNECT` ``, `` `TRACE` ``, or `` `TRACK` ``. [HTTPVERBSEC1], [HTTPVERBSEC2], [HTTPVERBSEC3]

To **normalize** a method, if it is a byte-case-insensitive match for `` `DELETE` ``, `` `GET` ``, `` `HEAD` ``, `` `OPTIONS` ``, `` `POST` ``, or `` `PUT` ``, byte-uppercase it.

Note

Normalization is done for backwards compatibility and consistency across APIs as methods are actually "case-sensitive".

Example

Using `` `patch` `` is highly likely to result in a `` `405 Method Not Allowed` ``. `` `PATCH` `` is much more likely to succeed.

Note

There are no restrictions on methods. `` `CHICKEN` `` is perfectly acceptable (and not a misspelling of `` `CHECKIN` ``). Other than those that are normalized there are no casing restrictions either. `` `Egg` `` or `` `eGg` `` would be fine, though uppercase is encouraged for consistency.

2.2.2. Headers §

A **header list** is a list of zero or more headers. It is initially the empty list.

Note

A header list is essentially a specialized multimap. An ordered list of key-value pairs with potentially duplicate keys.

A header list *list* **contains** a name *name* if *list* contains a header whose name is a byte-case-insensitive match for *name*.

To **get** a name *name* from a header list *list*, run these steps:

1. If *list* does not contain *name*, then return null.
2. Return the values of all headers in *list* whose name is a byte-case-insensitive match for *name*, separated from each other by 0x2C 0x20, in order.

To **get, decode, and split** a name *name* from header list *list*, run these steps:

1. Let *initialValue* be the result of getting *name* from *list*.
2. If *initialValue* is null, then return null.
3. Let *input* be the result of isomorphic decoding *initialValue*.
4. Let *position* be a position variable for *input*, initially pointing at the start of *input*.
5. Let *values* be a list of strings, initially empty.
6. Let *value* be the empty string.
7. While *position* is not past the end of *input*:
    1. Append the result of collecting a sequence of code points that are not U+0022 (") or U+002C (,) from *input*, given *position*, to *value*.

        Note

        The result might be the empty string.

    2. If *position* is not past the end of *input*, then:
        1. If the code point at *position* within *input* is U+0022 ("), then:
            1. Append the result of collecting an HTTP quoted string from *input*, given *position*, to *value*.
            2. If *position* is not past the end of *input*, then continue.
        2. Otherwise:
            1. Assert: the code point at *position* within *input* is U+002C (,).
            2. Advance *position* by 1.

File an issue about the selected text P tab or space from the start and end of *value*.

4. Append *value* to *values*.

5. Set *value* to the empty string.

8. Return *values*.

Example

This is how get, decode, and split functions in practice with `A` as the *name* argument:

Headers (as on the network)	**Output**
`A: nosniff,`	« "nosniff", "" »

`A: nosniff`
`B: sniff`
`A:`	

`A: text/html;", x/x`	« "text/html;", x/x" »

`A: text/html;"`
`A: x/x`	

`A: x/x;test="hi",y/y`	« "x/x;test="hi"", "y/y" »

`A: x/x;test="hi"`
`C: **bingo**`
`A: y/y`	

`A: x / x,,,1`	« "x / x", "", "", "1" »

`A: x / x`
`A: ,`
`A: 1`	

`A: "1,2", 3`	« ""1,2"", "3" »

`A: "1,2"`
`D: 4`
`A: 3`	

To **append** a name/value *name*/*value* pair to a header list *list*, run these steps:

1. If *list* contains *name*, then set *name* to the first such header's name.

   Note

   This reuses the casing of the name of the header already in list, *if any. If there are multiple matched headers their names will all be identical.*

2. Append a new header whose name is *name* and value is *value* to *list*.

To **delete** a name *name* from a header list *list*, remove all headers whose name is a byte-case-insensitive match for *name* from *list*.

To **set** a name/value *name*/*value* pair in a header list *list*, run these steps:

1. If *list* contains *name*, then set the value of the first such header to *value* and remove the others.

2. Otherwise, append a new header whose name is *name* and value is *value* to *list*.

To **combine** a name/value *name*/*value* pair in a header list *list*, run these steps:

1. If *list* contains *name*, then set the value of the first such header to its value, followed by 0x2C 0x20, followed by *value*.

2. Otherwise, append a new header whose name is *name* and value is *value* to *list*.

Note

Combine is used by `XMLHttpRequest` and the WebSocket protocol handshake.

To **sort and combine** a header list *list*, run these steps:

File an issue about the selected text / list of name-value pairs with the key being the name and value the value.

2. Let *names* be all the names of the headers in *list*, byte-lowercased, with duplicates removed, and finally sorted lexicographically.
3. For each *name* in *names*:
    1. Let *value* be the result of getting *name* from *list*.
    2. Assert: *value* is not null.
    3. Append *name*-*value* to *headers*.
4. Return *headers*.

A **header** consists of a **name** and **value**.

A name is a byte sequence that matches the field-name token production.

A value is a byte sequence that matches the following conditions:

- Has no leading or trailing HTTP tab or space bytes.
- Contains no 0x00 (NUL) or HTTP newline bytes.

Note

The definition of value is not defined in terms of an HTTP token production as it is broken.

To **normalize** a *potentialValue*, remove any leading and trailing HTTP whitespace bytes from *potentialValue*.

To determine whether a header *header* is a **CORS-safelisted request-header**, run these steps:

1. Let *value* be *header*'s value.
2. Byte-lowercase *header*'s name and switch on the result:

   ↪ `` `accept` ``

   If *value* contains a CORS-unsafe request-header byte, then return false.

   ↪ `` `accept-language` ``
   ↪ `` `content-language` ``

   If *value* contains a byte that is not in the range 0x30 (0) to 0x39 (9), inclusive, is not in the range 0x41 (A) to 0x5A (Z), inclusive, is not in the range 0x61 (a) to 0x7A (z), inclusive, and is not 0x20 (SP), 0x2A (*), 0x2C (,), 0x2D (-), 0x2E (.), 0x3B (;), or 0x3D (=), then return false.

   ↪ `` `content-type` ``

   1. If *value* contains a CORS-unsafe request-header byte, then return false.
   2. Let *mimeType* be the result of parsing *value*.
   3. If *mimeType* is failure, then return false.
   4. If *mimeType*'s essence is not "`application/x-www-form-urlencoded`", "`multipart/form-data`", or "`text/plain`", then return false.

   ⚠Warning!

   This intentionally does not use extract a MIME type as that algorithm is rather forgiving and servers are not expected to implement it.

   ¶ Example

   If extract a MIME type were used the following request would not result in a CORS preflight and a naïve parser on the server might treat the request body as JSON:

   ```
   fetch("https://victim.example/naïve-endpoint", {
     method: "POST",
     headers: [
       ["Content-Type", "application/json"],
       ["Content-Type", "text/plain"]
   ```

File an issue about the selected text

```
          ],
          credentials: "include",
          body: JSON.stringify(exerciseForTheReader)
        });
```

↪ **Otherwise**

Return false.

3. If *value*'s length is greater than 128, then return false.

4. Return true.

Note

There are limited exceptions to the `Content-Type` header safelist, as documented in CORS protocol exceptions.

A **CORS-unsafe request-header byte** is a byte *byte* for which one of the following is true:

- *byte* is less than 0x20 and is not 0x09 HT
- *byte* is 0x22 ("), 0x28 (left parenthesis), 0x29 (right parenthesis), 0x3A (:), 0x3C (<), 0x3E (>), 0x3F (?), 0x40 (@), 0x5B ([), 0x5C (\), 0x5D (]), 0x7B ({), 0x7D (}), or 0x7F DEL.

The **CORS-unsafe request-header names**, given a header list *headers*, are determined as follows:

1. Let *unsafeNames* be a new list.

2. Let *potentiallyUnsafeNames* be a new list.

3. Let *safelistValueSize* be 0.

4. For each *header* of *headers*:

   1. If *header* is not a CORS-safelisted request-header, then append *header*'s name to *unsafeNames*.

   2. Otherwise, append *header*'s name to *potentiallyUnsafeNames* and increase *safelistValueSize* by *header*'s value's length.

5. If *safelistValueSize* is greater than 1024, then for each *name* of *potentiallyUnsafeNames*, append *name* to *unsafeNames*.

6. Return *unsafeNames*, byte-lowercased, excluding duplicates, and sorted lexicographically.

A **CORS non-wildcard request-header name** is a byte-case-insensitive match for `Authorization`.

A **privileged no-CORS request-header name** is a header name that is a byte-case-insensitive match for one of

- `Range`.

Note

These are headers that can be set by privileged APIs, and will be preserved if their associated request object is copied, but will be removed if the request is modified by unprivilaged APIs.

`Range` headers are commonly used by downloads and media fetches, although neither of these currently specify how. html/2914 aims to solve this.

A helper is provided to add a range header to a particular request.

A **CORS-safelisted response-header name**, given a CORS-exposed header-name list *list*, is a header name that is a byte-case-insensitive match for one of

- `Cache-Control`
- `Content-Language`
- `Content-Length`
- `Content-Type`
- `Expires`
- `Last-Modified`
- `Pragma`
- Any value in *list* that is not a forbidden response-header name.

A **no-CORS-safelisted request-header name** is a header name that is a byte-case-insensitive match for one of

- `Accept`
- `Accept-Language`
- `Content-Language`
- `Content-Type`

To determine whether a header *header* is a **no-CORS-safelisted request-header**, run these steps:

File an issue about the selected text

1. If *header*'s name is not a no-CORS-safelisted request-header name, then return false.

2. Return whether *header* is a CORS-safelisted request-header.

A **forbidden header name** is a header name that is a byte-case-insensitive match for one of

- `` `Accept-Charset` ``
- `` `Accept-Encoding` ``
- `` `Access-Control-Request-Headers` ``
- `` `Access-Control-Request-Method` ``
- `` `Connection` ``
- `` `Content-Length` ``
- `` `Cookie` ``
- `` `Cookie2` ``
- `` `Date` ``
- `` `DNT` ``
- `` `Expect` ``
- `` `Host` ``
- `` `Keep-Alive` ``
- `` `Origin` ``
- `` `Referer` ``
- `` `TE` ``
- `` `Trailer` ``
- `` `Transfer-Encoding` ``
- `` `Upgrade` ``
- `` `Via` ``

or a header name that starts with a byte-case-insensitive match for `` `Proxy-` `` or `` `Sec-` `` (including being a byte-case-insensitive match for just `` `Proxy-` `` or `` `Sec-` ``).

Note

These are forbidden so the user agent remains in full control over them. Names starting with `` `Sec-` `` are reserved to allow new headers to be minted that are safe from APIs using fetch that allow control over headers by developers, such as `XMLHttpRequest`. [XHR]

A **forbidden response-header name** is a header name that is a byte-case-insensitive match for one of:

- `` `Set-Cookie` ``
- `` `Set-Cookie2` ``

To **extract header values** given a header *header*, run these steps:

1. If parsing *header*'s value, per the ABNF for *header*'s name, fails, then return failure.

2. Return one or more values resulting from parsing *header*'s value, per the ABNF for *header*'s name.

To **extract header list values** given a name *name* and a header list *list*, run these steps:

1. If *list* does not contain *name*, then return null.

2. If the ABNF for *name* allows a single header and *list* contains more than one, then return failure.

   Note

   If different error handling is needed, extract the desired header first.

3. Let *values* be an empty list.

4. For each header *header list* contains whose name is *name*:

   1. Let *extract* be the result of extracting header values from *header*.

   2. If *extract* is failure, then return failure.

   3. Append each value in *extract*, in order, to *values*.

5. Return *values*.

A **default `` `User-Agent` `` value** is a user-agent-defined value for the `` `User-Agent` `` header.

File an issue about the selected text

A **status** is a code.

A **null body status** is a status that is `101`, `204`, `205`, or `304`.

An **ok status** is any status in the range `200` to `299`, inclusive.

A **redirect status** is a status that is `301`, `302`, `303`, `307`, or `308`.

2.2.4. Bodies §

A **body** consists of:

- A **stream** (null or a `ReadableStream` object).
- A **transmitted bytes** (an integer), initially 0.
- A **total bytes** (an integer), initially 0.
- A **source**, initially null.

A body *body* is said to be **done** if *body* is null or *body*'s stream is closed or errored.

To **wait** for a body *body*, wait for *body* to be done.

To **clone** a body *body*, run these steps:

1. Let «*out1*, *out2*» be the result of teeing *body*'s stream.
2. Set *body*'s stream to *out1*.
3. Return a body whose stream is *out2* and other members are copied from *body*.

To **handle content codings** given *codings* and *bytes*, run these steps:

1. If *codings* are not supported, then return *bytes*.
2. Return the result of decoding *bytes* with *codings* as explained in HTTP, if decoding does not result in an error, and failure otherwise. [HTTP] [HTTP-SEMANTICS]

2.2.5. Requests §

The input to fetch is a **request**.

A request has an associated **method** (a method). Unless stated otherwise it is `` `GET` ``.

Note

This can be updated during redirects to `` `GET` `` as described in HTTP fetch.

A request has an associated **URL** (a URL).

Note

Implementations are encouraged to make this a pointer to the first URL in request's URL list. It is provided as a distinct field solely for the convenience of other standards hooking into Fetch.

A request has an associated **local-URLs-only flag**. Unless stated otherwise it is unset.

A request has an associated **header list** (a header list). Unless stated otherwise it is empty.

A request has an associated **unsafe-request flag**. Unless stated otherwise it is unset.

Note

The unsafe-request flag is set by APIs such as `fetch()` and `XMLHttpRequest` to ensure a CORS-preflight fetch is done based on the supplied method and header list. It does not free an API from outlawing forbidden methods and forbidden header names.

A request has an associated **body** (null or a body). Unless stated otherwise it is null.

File an issue about the selected text

Note

This can be updated during redirects to null as described in HTTP fetch.

A request has an associated **client** (null or an environment settings object).

A request has an associated **reserved client** (null, an environment, or an environment settings object). Unless stated otherwise it is null.

Note

This is only used by navigation requests and worker requests, but not service worker requests. It references an environment for a navigation request and an environment settings object for a worker request.

A request has an associated **replaces client id** (a string). Unless stated otherwise it is the empty string.

Note

This is only used by navigation requests. It is the id of the target browsing context's active document's environment settings object.

A request has an associated **window** ("`no-window`", "`client`", or an environment settings object whose global object is a `Window` object). Unless stated otherwise it is "`client`".

Note

The "`client`" value is changed to "`no-window`" or request's client during fetching. It provides a convenient way for standards to not have to explicitly set request's window.

A request has an associated **keepalive flag**. Unless stated otherwise it is unset.

Note

This can be used to allow the request to outlive the environment settings object, e.g., `navigator.sendBeacon` and the HTML `img` element set this flag. Requests with this flag set are subject to additional processing requirements.

A request has an associated **service-workers mode**, that is "`all`" or "`none`". Unless stated otherwise it is "`all`".

Note

This determines which service workers will receive a `fetch` event for this fetch.

"`all`"
Relevant service workers will get a `fetch` event for this fetch.

"`none`"
No service workers will get events for this fetch.

A request has an associated **initiator**, which is the empty string, "`download`", "`imageset`", "`manifest`", "`prefetch`", "`prerender`", or "`xslt`". Unless stated otherwise it is the empty string.

Note

A request's initiator is not particularly granular for the time being as other specifications do not require it to be. It is primarily a specification device to assist defining CSP and Mixed Content. It is not exposed to JavaScript. [CSP] [MIX]

A request has an associated **destination**, which is the empty string, "`audio`", "`audioworklet`", "`document`", "`embed`", "`font`", "`image`", "`manifest`", "`object`", "`paintworklet`", "`report`", "`script`", "`serviceworker`", "`sharedworker`", "`style`", "`track`", "`video`", "`worker`", or "`xslt`". Unless stated otherwise it is the empty string.

A request's destination is **script-like** if it is "`audioworklet`", "`paintworklet`", "`script`", "`serviceworker`", "`sharedworker`", or "`worker`".

⚠Warning!

Algorithms that use script-like should also consider "`xslt`" as that too can cause script execution. It is not included in the list as it is not always relevant and might require different behavior.

Note

The following table illustrates the relationship between a request's initiator, destination, CSP directives, and features. It is not exhaustive with respect to features. Features need to have the relevant values defined in their respective standards.

Initiator	*Destination*	*CSP directive*	*Features*
File an issue about the selected text		—	*CSP, NEL reports.*

	"document"		HTML's navigate algorithm.
	"document"	`child-src`	HTML's `<iframe>` and `<frame>`
	""	`connect-src`	`navigator.sendBeacon()`, `EventSource`, HTML's `<a ping="">` and `<area ping="">`, `fetch()`, `XMLHttpRequest`, `WebSocket`, Cache API
	"object"	`object-src`	HTML's `<object>`
	"embed"	`object-src`	HTML's `<embed>`
	"audio"	`media-src`	HTML's `<audio>`
	"font"	`font-src`	CSS' `@font-face`
	"image"	`img-src`	HTML's `<img src>`, `/favicon.ico` resource, SVG's `<image>`, CSS' `background-image`, CSS' `cursor`, CSS' `list-style-image`, …
	"audioworklet"	`script-src`	`audioWorklet.addModule()`
	"paintworklet"	`script-src`	`CSS.paintWorklet.addModule()`
	"script"	`script-src`	HTML's `<script>`, `importScripts()`
	"serviceworker"	`child-src`, `script-src`, `worker-src`	`navigator.serviceWorker.register()`
	"sharedworker"	`child-src`, `script-src`, `worker-src`	`SharedWorker`
	"worker"	`child-src`, `script-src`, `worker-src`	`Worker`
	"style"	`style-src`	HTML's `<link rel=stylesheet>`, CSS' `@import`
	"track"	`media-src`	HTML's `<track>`
	"video"	`media-src`	HTML's `<video>` element
"download"	""	—	HTML's `download=""`, "Save Link As…" UI
"imageset"	"image"	`img-src`	HTML's `<img srcset>` and `<picture>`
"manifest"	"manifest"	`manifest-src`	HTML's `<link rel=manifest>`
"prefetch"	""	`prefetch-src`	HTML's `<link rel=prefetch>`
"prerender"			HTML's `<link rel=prerender>`
"xslt"	"xslt"	`script-src`	`<?xml-stylesheet>`

CSP's `form-action` needs to be a hook directly in HTML's navigate or form submission algorithm.

CSP will also need to check request's client's responsible browsing context's ancestor browsing contexts for various CSP directives.

A request has an associated **priority** (null or a user-agent-defined object). Unless otherwise stated it is null.

A request has an associated **origin**, which is "`client`" or an origin. Unless stated otherwise it is "`client`".

Note

"`client`" is changed to an origin during fetching. It provides a convenient way for standards to not have to set request's origin.

A request has an associated **referrer**, which is "`no-referrer`", "`client`", or a URL. Unless stated otherwise it is "`client`".

Note

"`client`" is changed to "`no-referrer`" or a URL during fetching. It provides a convenient way for standards to not have to set request's referrer.

A request has an associated **referrer policy**, which is a referrer policy. Unless stated otherwise it is the empty string. [REFERRER]

Note

This can be used to override a referrer policy associated with an environment settings object.

A request has an associated **synchronous flag**. Unless stated otherwise it is unset.

A request has an associated **mode**, which is "`same-origin`", "`cors`", "`no-cors`", "`navigate`", or "`websocket`". Unless stated otherwise, it is "`no-cors`".

Note

"`same-origin`"

Used to ensure requests are made to same-origin URLs. Fetch will return a network error if the request is not made to a same-origin URL.

"`cors`"

Makes the request a CORS request. Fetch will return a network error if the requested resource does not understand the CORS protocol.

"`no-cors`"

File an issue about the selected text

Restricts requests to using CORS-safelisted methods and CORS-safelisted request-headers. Upon success, fetch will return an opaque filtered response.

"navigate"

This is a special mode used only when navigating between documents.

"websocket"

This is a special mode used only when establishing a WebSocket connection.

Even though the default request mode is "`no-cors`", standards are highly discouraged from using it for new features. It is rather unsafe.

A request has an associated **use-CORS-preflight flag**. Unless stated otherwise, it is unset.

Note

The use-CORS-preflight flag being set is one of several conditions that results in a CORS-preflight request. The use-CORS-preflight flag is set if either one or more event listeners are registered on an `XMLHttpRequestUpload` object or if a `ReadableStream` object is used in a request.

A request has an associated **credentials mode**, which is "`omit`", "`same-origin`", or "`include`". Unless stated otherwise, it is "`omit`".

Note

"omit"

Excludes credentials from this request.

"same-origin"

Include credentials with requests made to same-origin URLs.

"include"

Always includes credentials with this request.

Request's credentials mode controls the flow of credentials during a fetch. When request's mode is "`navigate`", its credentials mode is assumed to be "`include`" and fetch does not currently account for other values. If HTML changes here, this standard will need corresponding changes.

A request has an associated **use-URL-credentials flag**. Unless stated otherwise, it is unset.

A request has an associated **cache mode**, which is "`default`", "`no-store`", "`reload`", "`no-cache`", "`force-cache`", or "`only-if-cached`". Unless stated otherwise, it is "`default`".

Note

"default"

Fetch will inspect the HTTP cache on the way to the network. If there is a fresh response it will be used. If there is a stale response a conditional request will be created, and a normal request otherwise. It then updates the HTTP cache with the response. [HTTP] [HTTP-SEMANTICS] [HTTP-COND] [HTTP-CACHING] [HTTP-AUTH]

"no-store"

Fetch behaves as if there is no HTTP cache at all.

"reload"

Fetch behaves as if there is no HTTP cache on the way to the network. Ergo, it creates a normal request and updates the HTTP cache with the response.

"no-cache"

Fetch creates a conditional request if there is a response in the HTTP cache and a normal request otherwise. It then updates the HTTP cache with the response.

"force-cache"

Fetch uses any response in the HTTP cache matching the request, not paying attention to staleness. If there was no response, it creates a normal request and updates the HTTP cache with the response.

"only-if-cached"

Fetch uses any response in the HTTP cache matching the request, not paying attention to staleness. If there was no response, it returns a network error. (Can only be used when request's mode is "`same-origin`". Any cached redirects will be followed assuming request's redirect mode is "`follow`" and the redirects do not violate request's mode.)

If header list contains `` `If-Modified-Since` ``, `` `If-None-Match` ``, `` `If-Unmodified-Since` ``, `` `If-Match` ``, or `` `If-Range` ``, fetch will set cache mode to "`no-store`" if it is "`default`".

A request has an associated **redirect mode**, which is "`follow`", "`error`", or "`manual`". Unless stated otherwise, it is "`follow`".

File an issue about the selected text

Note

"follow"
: *Follow all redirects incurred when fetching a resource.*

"error"
: *Return a network error when a request is met with a redirect.*

"manual"
: *Retrieves an opaque-redirect filtered response when a request is met with a redirect so that the redirect can be followed manually.*

A request has associated **integrity metadata** (a string). Unless stated otherwise, it is the empty string.

A request has associated **cryptographic nonce metadata** (a string). Unless stated otherwise, it is the empty string.

A request has associated **parser metadata** which is the empty string, "`parser-inserted`", or "`not-parser-inserted`". Unless otherwise stated, it is the empty string.

Note

A request's cryptographic nonce metadata and parser metadata are generally populated from attributes and flags on the HTML element responsible for creating a request. They are used by various algorithms in Content Security Policy to determine whether requests or responses are to be blocked in a given context. [CSP]

A request has an associated **reload-navigation flag**. Unless stated otherwise, it is unset.

Note

This flag is for exclusive use by HTML's navigate algorithm. [HTML]

A request has an associated **history-navigation flag**. Unless stated otherwise, it is unset.

Note

This flag is for exclusive use by HTML's navigate algorithm. [HTML]

A request has an associated **tainted origin flag**. Unless stated otherwise, it is unset.

A request has an associated **URL list** (a list of one or more URLs). Unless stated otherwise, it is a list containing a copy of request's URL.

A request has an associated **current URL**. It is a pointer to the last URL in request's URL list.

A request has an associated **redirect count**. Unless stated otherwise, it is zero.

A request has an associated **response tainting**, which is "`basic`", "`cors`", or "`opaque`". Unless stated otherwise, it is "`basic`".

A request has an associated **done flag**. Unless stated otherwise, it is unset.

Note

A request's tainted origin flag, URL list, current URL, redirect count, response tainting, and done flag are used as bookkeeping details by the fetch algorithm.

A **subresource request** is a request whose destination is "`audio`", "`audioworklet`", "`font`", "`image`", "`manifest`", "`paintworklet`", "`script`", "`style`", "`track`", "`video`", "`xslt`", or the empty string.

A **potential-navigation-or-subresource request** is a request whose destination is "`object`" or "`embed`".

A **non-subresource request** is a request whose destination is "`document`", "`report`", "`serviceworker`", "`sharedworker`", or "`worker`".

A **navigation request** is a request whose destination is "`document`".

Note

See handle fetch for usage of these terms. [SW]

File an issue about the selected text

Serializing a request origin, given a request *request*, is to run these steps:

1. If *request*'s tainted origin flag is set, then return `` `null` ``.
2. Return *request*'s origin, serialized and isomorphic encoded.

To **clone** a request *request*, run these steps:

1. Let *newRequest* be a copy of *request*, except for its body.
2. If *request*'s body is non-null, set *newRequest*'s body to the result of cloning *request*'s body.
3. Return *newRequest*.

To **transmit body** for a request *request*, run these steps:

1. Let *body* be *request*'s body.
2. If *body* is null, then queue a fetch task on *request* to process request end-of-body for *request* and abort these steps.
3. Let *reader* be the result of getting a reader from *body*'s stream.

   Note

   This operation cannot throw an exception.

4. Let *read* be the result of reading a chunk from *body*'s stream with *reader*.
5. In parallel, while true:
   1. Run these steps, but abort when the ongoing fetch is terminated:
      1. Wait for *read* to be fulfilled or rejected.
      2. If *read* is fulfilled with an object whose `done` property is false and whose `value` property is a `Uint8Array` object, then run these steps:
         1. Let *bs* be the byte sequence represented by the `Uint8Array` object.
         2. Transmit *bs*. Whenever one or more bytes are transmitted, increase *body*'s transmitted bytes by the number of transmitted bytes and queue a fetch task on *request* to process request body for *request*.

            Note

            This step blocks until bs *is fully transmitted.*

         3. Set *read* to the result of reading a chunk from *body*'s stream with *reader*.
      3. Otherwise, if *read* is fulfilled with an object whose `done` property is true, then queue a fetch task on *request* to process request end-of-body for *request* and abort these in-parallel steps.
      4. Otherwise, if *read* is rejected with an "`AbortError`" `DOMException`, terminate the ongoing fetch with the aborted flag set.
      5. Otherwise, terminate the ongoing fetch.
   2. If aborted, then abort these in-parallel steps.

To **add a range header** to a request *request*, with an integer *first*, and an optional integer *last*, run these steps:

1. Let *rangeValue* be `` `bytes ` ``.
2. Serialize and isomorphic encode *first*, and append the result to *rangeValue*.
3. Append 0x2D (-) to *rangeValue*.
4. If *last* is given, then serialize and isomorphic encode it, and append the result to *rangeValue*.

File an issue about the selected text

5. Append `Range`/*rangeValue* to *request*'s header list.

Note

A range header denotes an inclusive byte range. There a range header where first *is 0 and* last *is 500, is a range of 501 bytes.*

Note

Features that combine multiple responses into one logical resource are historically a source of security bugs. Please seek security review for features that deal with partial responses.

2.2.6. Responses §

The result of fetch is a **response**. A response evolves over time. That is, not all its fields are available straight away.

A response has an associated **type** which is "`basic`", "`cors`", "`default`", "`error`", "`opaque`", or "`opaqueredirect`". Unless stated otherwise, it is "`default`".

A response can have an associated **aborted flag**, which is initially unset.

Note

This indicates that the request was intentionally aborted by the developer or end-user.

A response has an associated **URL**. It is a pointer to the last response URL in response's URL list and null if response's URL list is the empty list.

A response has an associated **URL list** (a list of zero or more response URLs). Unless stated otherwise, it is the empty list.

Note

Except for the last response URL, if any, a response's URL list cannot be exposed to script. That would violate atomic HTTP redirect handling.

A response has an associated **status**, which is a status. Unless stated otherwise it is `200`.

A response has an associated **status message**. Unless stated otherwise it is the empty byte sequence.

Note

Responses over an HTTP/2 connection will always have the empty byte sequence as status message as HTTP/2 does not support them.

A response has an associated **header list** (a header list). Unless stated otherwise it is empty.

A response has an associated **body** (null or a body). Unless stated otherwise it is null.

A response has an associated **trailer** (a header list). Unless stated otherwise it is empty.

A response has an associated **trailer failed flag**, which is initially unset.

A response has an associated **cache state** (the empty string or "`local`"). Unlesss stated otherwise, it is the empty string.

Note

This is intended solely for usage by service workers. [SW]

A response has an associated **HTTPS state** (an HTTPS state value). Unless stated otherwise, it is "`none`".

A response has an associated **CSP list**, which is a list of Content Security Policy objects for the response. The list is empty unless otherwise specified. [CSP]

A response has an associated **CORS-exposed header-name list** (a list of zero or more header names). The list is empty unless otherwise specified.

Note

A response will typically get its CORS-exposed header-name list set by extracting header values from the `Access-Control-Expose-Headers` header. This list is used by a CORS filtered response to determine which headers to expose.

A response has an associated **range-requested flag**, which is initially unset.

Note

This is used to ensure to prevent a partial response from an earlier ranged request being provided to an API that didn't make a range request. See the flag's usage for a detailed description of the attack.

A response can have an associated **location URL** (null, failure, or a URL). Unless specified otherwise, response has no location URL.

File an issue about the selected text

Note

This concept is used for redirect handling in Fetch and in HTML's navigate algorithm. It ensures `Location` has its value extracted consistently and only once. [HTML]

A response whose type is "`error`" and aborted flag is set is known as an **aborted network error**.

A response whose type is "`error`" is known as a **network error**.

A network error is a response whose status is always `0`, status message is always the empty byte sequence, header list is always empty, body is always null, and trailer is always empty.

A **filtered response** is a limited view on a response that is not a network error. This response is referred to as the filtered response's associated **internal response**.

Note

The fetch algorithm returns such a view to ensure APIs do not accidentally leak information. If the information needs to be exposed for legacy reasons, e.g., to feed image data to a decoder, the associated internal response can be used, which is only "accessible" to internal specification algorithms and is never a filtered response itself.

A **basic filtered response** is a filtered response whose type is "`basic`" and header list excludes any headers in internal response's header list whose name is a forbidden response-header name.

A **CORS filtered response** is a filtered response whose type is "`cors`", header list excludes any headers in internal response's header list whose name is *not* a CORS-safelisted response-header name, given internal response's CORS-exposed header-name list, and trailer is empty.

An **opaque filtered response** is a filtered response whose type is "`opaque`", URL list is the empty list, status is `0`, status message is the empty byte sequence, header list is empty, body is null, and trailer is empty.

An **opaque-redirect filtered response** is a filtered response whose type is "`opaqueredirect`", status is `0`, status message is the empty byte sequence, header list is empty, body is null, and trailer is empty.

Note

Exposing the URL list for opaque-redirect filtered responses is harmless since no redirects are followed.

In other words, an opaque filtered response and an opaque-redirect filtered response are nearly indistinguishable from a network error. When introducing new APIs, do not use the internal response for internal specification algorithms as that will leak information.

This also means that JavaScript APIs, such as `response.ok`, will return rather useless results.

To **clone** a response *response*, run these steps:

1. If *response* is a filtered response, then return a new identical filtered response whose internal response is a clone of *response*'s internal response.
2. Let *newResponse* be a copy of *response*, except for its body.
3. If *response*'s body is non-null, then set *newResponse*'s body to the result of cloning *response*'s body.
4. Return *newResponse*.

2.2.7. Miscellaneous §

A **potential destination** is "`fetch`" or a destination which is not the empty string.

To **translate** a potential destination *potentialDestination*, run these steps:

1. If *potentialDestination* is "`fetch`", then return the empty string.
2. Assert: *potentialDestination* is a destination.

File an issue about the selected text *ion*.

2.3. Authentication entries §

An **authentication entry** and a **proxy-authentication entry** are tuples of username, password, and realm, used for HTTP authentication and HTTP proxy authentication, and associated with one or more requests.

User agents should allow both to be cleared together with HTTP cookies and similar tracking functionality.

Further details are defined by HTTP. [HTTP] [HTTP-SEMANTICS] [HTTP-COND] [HTTP-CACHING] [HTTP-AUTH]

2.4. Fetch groups §

Each environment settings object has an associated **fetch group**.

A fetch group holds an ordered list of **fetch records**.

A fetch record has an associated **request** (a request).

A fetch record has an associated **fetch** (a fetch algorithm or null).

When a fetch group is **terminated**, for each associated fetch record whose request's done flag or keepalive flag is unset, terminate the fetch record's fetch.

2.5. Connections §

A user agent has an associated **connection pool**. A connection pool consists of zero or more **connections**. Each connection is identified by an **origin** (an origin) and **credentials** (a boolean).

To **obtain a connection**, given an *origin* and *credentials*, run these steps:

1. If connection pool contains a connection whose **origin** is *origin* and **credentials** is *credentials*, then return that connection.
2. Let *connection* be null.
3. Run these steps, but abort when the ongoing fetch is terminated:
    1. Set *connection* to the result of establishing an HTTP connection to *origin*. [HTTP] [HTTP-SEMANTICS] [HTTP-COND] [HTTP-CACHING] [HTTP-AUTH] [TLS]

       If *credentials* is false, then do *not* send a TLS client certificate.

       If establishing a connection does not succeed (e.g., a DNS, TCP, or TLS error), then return failure.
4. If aborted, then:
    1. If *connection* is not null, then close *connection*.
    2. Return failure.
5. Add *connection* to the connection pool with **origin** being *origin* and **credentials** being *credentials*.
6. Return *connection*.

Note

*This is intentionally a little vague as the finer points are still evolving. Describing this helps explain the `<link rel=preconnect>` feature and clearly stipulates that connections are keyed on **credentials**. The latter clarifies that e.g., TLS session identifiers are not reused across connections whose **credentials** are false with connections whose **credentials** are true.*

2.6. Port blocking §

To determine whether fetching a request *request* **should be blocked due to a bad port**, run these steps:

File an issue about the selected text

1. Let *url* be *request*'s current URL.
2. Let *scheme* be *url*'s scheme.
3. Let *port* be *url*'s port.
4. If *scheme* is "`ftp`" and *port* is 20 or 21, then return **allowed**.
5. Otherwise, if *scheme* is a network scheme and *port* is a bad port, then return **blocked**.
6. Return **allowed**.

A port is a **bad port** if it is listed in the first column of the following table.

Port	Typical service
1	tcpmux
7	echo
9	discard
11	systat
13	daytime
15	netstat
17	qotd
19	chargen
20	ftp-data
21	ftp
22	ssh
23	telnet
25	smtp
37	time
42	name
43	nicname
53	domain
77	priv-rjs
79	finger
87	ttylink
95	supdup
101	hostriame
102	iso-tsap
103	gppitnp
104	acr-nema
109	pop2
110	pop3
111	sunrpc
113	auth
115	sftp
117	uucp-path
119	nntp
123	ntp
135	loc-srv / epmap
139	netbios
143	imap2
179	bgp
389	ldap
427	afp (alternate)
465	smtp (alternate)
512	print / exec
513	login
514	shell
515	printer
526	tempo
530	courier
531	chat
532	netnews
540	uucp
548	afp
556	remotefs
563	nntp+ssl
587	smtp (outgoing)
601	syslog-conn
636	ldap+ssl

File an issue about the selected text

995	pop3+ssl
2049	nfs
3659	apple-sasl
4045	lockd
6000	x11
6665	irc (alternate)
6666	irc (alternate)
6667	irc (default)
6668	irc (alternate)
6669	irc (alternate)
6697	irc+tls

2.7. Should *response* to *request* be blocked due to its MIME type?

Run these steps:

1. Let *mimeType* be the result of extracting a MIME type from *response*'s header list.
2. If *mimeType* is failure, then return **allowed**.
3. Let *destination* be *request*'s destination.
4. If *destination* is script-like and one of the following is true, then return **blocked**:
   - *mimeType*'s essence starts with "`audio/`", "`image/`", or "`video/`".
   - *mimeType*'s essence is "`text/csv`".
5. Return **allowed**.

2.8. Streams §

Note

This section might be integrated into other standards, such as IDL.

2.8.1. ReadableStream §

A **`ReadableStream`** object represents a stream of data. In this section, we define common operations for `ReadableStream` objects. [STREAMS]

To **enqueue** *chunk* into a `ReadableStream` object *stream*, run these steps:

1. Call ReadableStreamDefaultControllerEnqueue(*stream*.[[readableStreamController]], *chunk*).

To **close** a `ReadableStream` object *stream*, run these steps:

1. Call ReadableStreamDefaultControllerClose(*stream*.[[readableStreamController]]).

To **error** a `ReadableStream` object *stream* with given *reason*, run these steps:

1. Call ReadableStreamDefaultControllerError(*stream*.[[readableStreamController]]). *reason*).

To **construct a `ReadableStream` object** optionally with a *highWaterMark*, *sizeAlgorithm* algorithm, *pull* action, and *cancel* action, run these steps:

Note

This algorithm used to take a strategy *parameter, whose `highWaterMark` and `sizeAlgorithm` members were extracted to provide what are now separate parameters. If another specification still passes that* strategy *parameter, please update it.*

1. Let *startAlgorithm* be an algorithm that returns undefined.
2. If *pull* is absent, then set it to an action that returns undefined.
3. Let *pullAlgorithm* be an algorithm that returns the result of promise-calling *pull*().
4. If *cancel* is absent, then set it to an action that returns undefined.

File an issue about the selected text ın algorithm that takes a *reason* and returns the result of promise-calling *cancel*(*reason*).

6. If *highWaterMark* is absent, then set it to 1.
7. If *sizeAlgorithm* is absent, then set it to an algorithm that returns 1.
8. Return CreateReadableStream(*startAlgorithm*, *pullAlgorithm*, *cancelAlgorithm*, *highWaterMark*, *sizeAlgorithm*).

To **construct a fixed `ReadableStream` object** with given *chunks*, run these steps:

1. Let *stream* be the result of constructing a `ReadableStream` object.
2. For each *chunk* in *chunks*, enqueue *chunk* into *stream*.
3. Close *stream*.
4. Return *stream*.

To **get a reader** from a `ReadableStream` object *stream*, run these steps:

1. Let *reader* be the result of calling AcquireReadableStreamDefaultReader(*stream*).
2. Return *reader*.

To **read a chunk** from a `ReadableStream` object with *reader*, return the result of calling ReadableStreamDefaultReaderRead(*reader*).

To **read all bytes** from a `ReadableStream` object with *reader*, run these steps:

1. Let *promise* be a new promise.
2. Let *bytes* be an empty byte sequence.
3. Let *read* be the result of calling ReadableStreamDefaultReaderRead(*reader*).
    - When *read* is fulfilled with an object whose `done` property is false and whose `value` property is a `Uint8Array` object, append the `value` property to *bytes* and run the above step again.
    - When *read* is fulfilled with an object whose `done` property is true, resolve *promise* with *bytes*.
    - When *read* is fulfilled with a value that matches with neither of the above patterns, reject *promise* with a `TypeError`.
    - When *read* is rejected with an error, reject *promise* with that error.
4. Return *promise*.

To **cancel** a `ReadableStream` object *stream* with *reason*, return the result of calling ReadableStreamCancel(*stream*, *reason*).

Note

Because the reader grants exclusive access, the actual mechanism of how to read cannot be observed. Implementations could use more direct mechanism if convenient.

To **tee** a `ReadableStream` object *stream*, run these steps:

1. Return the result of calling ReadableStreamTee(*stream*, true).

An **empty** `ReadableStream` object is the result of constructing a fixed `ReadableStream` object with an empty list.

Note

Constructing an empty `ReadableStream` object will not throw an exception.

A `ReadableStream` object *stream* is said to be **readable** if *stream*.[[state]] is "readable".

A `ReadableStream` object *stream* is said to be **closed** if *stream*.[[state]] is "closed".

A `ReadableStream` object *stream* is said to be **errored** if *stream*.[[state]] is "errored".

A `ReadableStream` object *stream* is said to be **locked** if the result of calling IsReadableStreamLocked(*stream*) is true.

A `ReadableStream` object *stream* is said to **need more data** if the following conditions hold:

- *stream* is readable.
- The result of calling ReadableStreamDefaultControllerGetDesiredSize(*stream*.[[readableStreamController]]). is positive.

A `ReadableStream` object *stream* is said to be **disturbed** if the result of calling IsReadableStreamDisturbed(*stream*) is true.

File an issue about the selected text

3. HTTP extensions §

3.1. `Origin` header §

The `**Origin**` request header indicates where a fetch originates from.

Note

The `Origin` header is a version of the `Referer` [sic] header that does not reveal a path. It is used for all HTTP fetches whose CORS flag *is set as well as those where request's method is neither `GET` nor `HEAD`. Due to compatibility constraints it is not included in all fetches.*

Its value ABNF:

```
Origin                           = origin-or-null

origin-or-null                   = origin / %s"null" ; case-sensitive
origin                           = scheme "://" host [ ":" port ]
```

Note

This supplants the `Origin` header. [ORIGIN]

3.2. CORS protocol §

To allow sharing responses cross-origin and allow for more versatile fetches than possible with HTML's `form` element, the **CORS protocol** exists. It is layered on top of HTTP and allows responses to declare they can be shared with other origins.

Note

It needs to be an opt-in mechanism to prevent leaking data from responses behind a firewall (intranets). Additionally, for requests including credentials it needs to be opt-in to prevent leaking potentially-sensitive data.

This section explains the CORS protocol as it pertains to server developers. Requirements for user agents are part of the fetch algorithm, except for the new HTTP header syntax.

3.2.1. General §

The CORS protocol consists of a set of headers that indicates whether a response can be shared cross-origin.

For requests that are more involved than what is possible with HTML's `form` element, a CORS-preflight request is performed, to ensure request's current URL supports the CORS protocol.

3.2.2. HTTP requests §

A **CORS request** is an HTTP request that includes an `Origin` header. It cannot be reliably identified as participating in the CORS protocol as the `Origin` header is also included for all requests whose method is neither `GET` nor `HEAD`.

A **CORS-preflight request** is a CORS request that checks to see if the CORS protocol is understood. It uses `OPTIONS` as method and includes these headers:

`***Access-Control-Request-Method***`
: Indicates which method a future CORS request to the same resource might use.

`***Access-Control-Request-Headers***`
: Indicates which headers a future CORS request to the same resource might use.

File an issue about the selected text

3.2.3. HTTP responses §

An HTTP response to a CORS request can include the following headers:

`Access-Control-Allow-Origin`
Indicates whether the response can be shared, via returning the literal value of the `Origin` request header (which can be `null`) or `*` in a response.

`Access-Control-Allow-Credentials`
Indicates whether the response can be shared when request's credentials mode is "include".

Note

For a CORS-preflight request, request's credentials mode is always "omit", but for any subsequent CORS requests it might not be. Support therefore needs to be indicated as part of the HTTP response to the CORS-preflight request as well.

An HTTP response to a CORS-preflight request can include the following headers:

`Access-Control-Allow-Methods`
Indicates which methods are supported by the response's URL for the purposes of the CORS protocol.

Note

The `Allow` header is not relevant for the purposes of the CORS protocol.

`Access-Control-Allow-Headers`
Indicates which headers are supported by the response's URL for the purposes of the CORS protocol.

`Access-Control-Max-Age`
Indicates how long the information provided by the `Access-Control-Allow-Methods` and `Access-Control-Allow-Headers` headers can be cached.

An HTTP response to a CORS request that is not a CORS-preflight request can also include the following header:

`Access-Control-Expose-Headers`
Indicates which headers can be exposed as part of the response by listing their names.

In case a server does not wish to participate in the CORS protocol, its HTTP response to the CORS or CORS-preflight request must not include any of the above headers. The server is encouraged to use the 403 status in such HTTP responses.

3.2.4. HTTP new-header syntax §

ABNF for the values of the headers used by the CORS protocol:

```
Access-Control-Request-Method    = method
Access-Control-Request-Headers   = 1#field-name

wildcard                         = "*"
Access-Control-Allow-Origin      = origin-or-null / wildcard
Access-Control-Allow-Credentials = %s"true" ; case-sensitive
Access-Control-Expose-Headers    = #field-name
Access-Control-Max-Age           = delta-seconds
Access-Control-Allow-Methods     = #method
Access-Control-Allow-Headers     = #field-name
```

Note

For `Access-Control-Expose-Headers`, `Access-Control-Allow-Methods`, and `Access-Control-Allow-Headers` response headers the value `*` counts as a wildcard for requests without credentials. For such requests there is no way to solely match a header name or method that is `*`.

3.2.5. CORS protocol and credentials §

File an issue about the selected text

When request's credentials mode is "include" it has an impact on the functioning of the CORS protocol other than including credentials in the fetch.

Example

In the old days, XMLHttpRequest could be used to set request's credentials mode to "include":

```
var client = new XMLHttpRequest()
client.open("GET", "./")
client.withCredentials = true
/* … */
```

Nowadays, `fetch("./", { credentials:"include" }).then(/* … */)` suffices.

A request's credentials mode is not necessarily observable on the server; only when credentials exist for a request can it be observed by virtue of the credentials being included. Note that even so, a CORS-preflight request never includes credentials.

The server developer therefore needs to decide whether or not responses "tainted" with credentials can be shared. And also needs to decide if requests necessitating a CORS-preflight request can include credentials. Generally speaking, both sharing responses and allowing requests with credentials is rather unsafe, and extreme care has to be taken to avoid the confused deputy problem.

To share responses with credentials, the `Access-Control-Allow-Origin` and `Access-Control-Allow-Credentials` headers are important. The following table serves to illustrate the various legal and illegal combinations for a request to `https://rabbit.invalid/`:

Request's credentials mode	`Access-Control-Allow-Origin`	`Access-Control-Allow-Credentials`	Shared?	Notes
"omit"	`*`	Omitted	✓	—
"omit"	`*`	`true`	✓	If credentials mode is not "include", then `Access-Control-Allow-Credentials` is ignored.
"omit"	`https://rabbit.invalid/`	Omitted	❌	A serialized origin has no trailing slash.
"omit"	`https://rabbit.invalid`	Omitted	✓	—
"include"	`*`	`true`	❌	If credentials mode is "include", then `Access-Control-Allow-Origin` cannot be `*`.
"include"	`https://rabbit.invalid`	`true`	✓	—
"include"	`https://rabbit.invalid`	`True`	❌	`true` is (byte) case-sensitive.

Similarly, `Access-Control-Expose-Headers`, `Access-Control-Allow-Methods`, and `Access-Control-Allow-Headers` response headers can only use `*` as value when request's credentials mode is not "include".

3.2.6. Examples §

Example

A script at `https://foo.invalid/` wants to fetch some data from `https://bar.invalid/`. (Neither credentials nor response header access is important.)

```
var url = "https://bar.invalid/api?
key=730d67a37d7f3d802e96396d00280768773813fbe726d116944d814422fc1a45&data=about:unicorn";
fetch(url).then(success, failure)
```

This will use the CORS protocol, though this is entirely transparent to the developer from `foo.invalid`. As part of the CORS protocol, the user agent will include the `Origin` header in the request:

```
Origin: https://foo.invalid
```

Upon receiving a response from `bar.invalid`, the user agent will verify the `Access-Control-Allow-Origin` response header. If its value is either `https://foo.invalid` or `*`, the user agent will invoke the `success` callback. If it has any other value, or is missing, the user agent will invoke the `failure` callback.

Example

The developer of `foo.invalid` is back, and now wants to fetch some data from `bar.invalid` while also accessing a response header.

```
fetch(url).then(response => {
  var hsts = response.headers.get("strict-transport-security"),
```

File an issue about the selected text

```
    csp = response.headers.get("content-security-policy")
  log(hsts, csp)
})
```

bar.invalid provides a correct `Access-Control-Allow-Origin` response header per the earlier example. The values of hsts and csp will depend on the `Access-Control-Expose-Headers` response header. For example, if the response included the following headers

```
Content-Security-Policy: default-src 'self'
Strict-Transport-Security: max-age=31536000; includeSubdomains; preload
Access-Control-Expose-Headers: Content-Security-Policy
```

then hsts would be null and csp would be "default-src 'self'", even though the response did include both headers. This is because bar.invalid needs to explicitly share each header by listing their names in the `Access-Control-Expose-Headers` response header.

Alternatively, if bar.invalid wanted to share all its response headers, for requests that do not include credentials, it could use `*` as value for the `Access-Control-Expose-Headers` response header. If the request would have included credentials, the response header names would have to be listed explicitly and `*` could not be used.

Example

The developer of foo.invalid returns, now fetching some data from bar.invalid while including credentials. This time around the CORS protocol is no longer transparent to the developer as credentials require an explicit opt-in:

```
fetch(url, { credentials:"include" }).then(success, failure)
```

This also makes any `Set-Cookie` response headers bar.invalid includes fully functional (they are ignored otherwise).

The user agent will make sure to include any relevant credentials in the request. It will also put stricter requirements on the response. Not only will bar.invalid need to list `https://foo.invalid` as value for the `Access-Control-Allow-Origin` header (`*` is not allowed when credentials are involved), the `Access-Control-Allow-Credentials` header has to be present too:

```
Access-Control-Allow-Origin: https://foo.invalid
Access-Control-Allow-Credentials: true
```

If the response does not include those two headers with those values, the failure callback will be invoked. However, any `Set-Cookie` response headers will be respected.

3.2.7. CORS protocol exceptions §

Specifications have allowed limited exceptions to the CORS safelist for non-safelisted `Content-Type` header values. These exceptions are made for requests that can be triggered by web content but whose headers and bodies can be only minimally controlled by the web content. Therefore, servers should expect cross-origin web content to be allowed to trigger non-preflighted requests with the following non-safelisted `Content-Type` header values:

- `application/csp-report` [CSP]
- `application/expect-ct-report+json` [EXPECT-CT]
- `application/xss-auditor-report`
- `application/ocsp-request` [OCSP]

Specifications should avoid introducing new exceptions and should only do so with careful consideration for the security consequences. New exceptions can be proposed by filing an issue.

3.3. `Content-Type` header §

The `Content-Type` header is largely defined in HTTP. Its processing model is defined here as the ABNF defined in HTTP is not compatible with web content. [HTTP]

To **extract a MIME type** from a header list *headers*, run these steps:

1. Let *charset* be null.

File an issue about the selected text

3. Let *mimeType* be null.
4. Let *values* be the result of getting, decoding, and splitting `Content-Type` from *headers*.
5. If *values* is null, then return failure.
6. For each *value* of *values*:
    1. Let *temporaryMimeType* be the result of parsing *value*.
    2. If *temporaryMimeType* is failure or its essence is "*/*", then continue.
    3. Set *mimeType* to *temporaryMimeType*.
    4. If *mimeType*'s essence is not *essence*, then:
        1. Set *charset* to null.
        2. If *mimeType*'s parameters["charset"] exists, then set *charset* to *mimeType*'s parameters["charset"].
        3. Set *essence* to *mimeType*'s essence.
    5. Otherwise, if *mimeType*'s parameters["charset"] does not exist, and *charset* is non-null, set *mimeType*'s parameters["charset"] to *charset*.
7. If *mimeType* is null, then return failure.
8. Return *mimeType*.

⚠Warning!

When extract a MIME type returns failure or a MIME type whose essence is incorrect for a given format, treat this as a fatal error. Existing web platform features have not always followed this pattern, which has been a major source of security vulnerabilities in those features over the years. In contrast, a MIME type's parameters can typically be safely ignored.

Example

This is how extract a MIME type functions in practice:

Headers (as on the network)	Output (serialized)
`Content-Type: text/plain;charset=gbk, text/html`	`text/html`
`Content-Type: text/html;charset=gbk;a=b, text/html;x=y`	`text/html;x=y;charset=gbk`

`Content-Type: text/html;charset=gbk;a=b`
`Content-Type: text/html;x=y`	

`Content-Type: text/html;charset=gbk`
`Content-Type: x/x`
`Content-Type: text/html;x=y`
	`text/html;x=y`

`Content-Type: text/html`
`Content-Type: cannot-parse`
	`text/html`

`Content-Type: text/html`
`Content-Type: */*`	

`Content-Type: text/html`
`Content-Type:`	

3.4. `X-Content-Type-Options` header §

The `X-Content-Type-Options` response header can be used to require checking of a response's `Content-Type` header against the destination of a request.

To **determine nosniff**, given a header list *list*, run these steps:

1. Let *values* be the result of getting, decoding, and splitting `X-Content-Type-Options` from *list*.

File an issue about the selected text ırn false.

3. If *values*[0] is an ASCII case-insensitive match for "`nosniff`", then return true.

4. Return false.

Web developers and conformance checkers must use the following value ABNF for `` `X-Content-Type-Options` ``:

```
X-Content-Type-Options           = "nosniff" ; case-insensitive
```

3.4.1. Should *response* to *request* be blocked due to nosniff?

Run these steps:

1. If determine nosniff with *response*'s header list is false, then return **allowed**.

2. Let *mimeType* be the result of extracting a MIME type from *response*'s header list.

3. Let *destination* be *request*'s destination.

4. If *destination* is script-like and *mimeType* is failure or is not a JavaScript MIME type, then return **blocked**.

5. If *destination* is "`style`" and *mimeType* is failure or its essence is not "`text/css`", then return **blocked**.

6. Return **allowed**.

Note

Only request destinations that are script-like or "`style`" are considered as any exploits pertain to them. Also, considering "`image`" was not compatible with deployed content.

3.5. CORB §

Note

Cross-origin read blocking, better known as CORB, is an algorithm which identifies dubious cross-origin resource fetches (e.g., fetches that would fail anyway like attempts to render JSON inside an `img` element) and blocks them before they reach a web page. CORB reduces the risk of leaking sensitive data by keeping it further from cross-origin web pages.

A **CORB-protected MIME type** is an HTML MIME type, a JSON MIME type, or an XML MIME type excluding `image/svg+xml`.

Note

Even without CORB, accessing the content of cross-origin resources with CORB-protected MIME types is either managed by the CORS protocol (e.g., in case of `XMLHttpRequest`), not observable (e.g., in case of pings or CSP reports which ignore the response), or would result in an error (e.g., when failing to decode an HTML document embedded in an `img` element as an image). This means that CORB can block CORB-protected MIME types resources without being disruptive to web pages.

To perform a **CORB check**, given a *request* and *response*, run these steps:

1. If *request*'s initiator is "`download`", then return **allowed**.

   If we recast downloading as navigation this step can be removed.

2. If *request*'s current URL's scheme is not an HTTP(S) scheme, then return **allowed**.

3. Let *mimeType* be the result of extracting a MIME type from *response*'s header list.

4. If *mimeType* is failure, then return **allowed**.

5. If *response*'s status is `206` and *mimeType* is a CORB-protected MIME type, then return **blocked**.

6. If determine nosniff with *response*'s header list is true and *mimeType* is a CORB-protected MIME type or its essence is "`text/plain`", then return **blocked**.

   Note

   CORB only protects `text/plain` responses with a `` `X-Content-Type-Options: nosniff` `` header. Unfortunately, protecting such responses without that header when their status is `206` would break too many existing video responses that have a `text/plain` MIME type.

File an issue about the selected text

7. Return **allowed**.

3.6. `` `Cross-Origin-Resource-Policy` `` header §

The `` `Cross-Origin-Resource-Policy` `` response header can be used to require checking a request's current URL's origin against a request's origin when request's mode is "`no-cors`".

Its value ABNF:

```
Cross-Origin-Resource-Policy     = %s"same-origin" / %s"same-site" ; case-sensitive
```

To perform a **cross-origin resource policy check**, given a *request* and *response*, run these steps:

1. If *request*'s mode is not "`no-cors`", then return **allowed**.
2. If *request*'s origin is same origin with *request*'s current URL's origin, then return **allowed**.

   Note

   While redirects that carry a `` `Cross-Origin-Resource-Policy` `` header are checked, redirects without such a header resulting in response *do not contribute to this algorithm. I.e.,* request*'s tainted origin flag is not checked.*

3. Let *policy* be the result of getting `` `Cross-Origin-Resource-Policy` `` from *response*'s header list.

   Note

   *This means that `` `Cross-Origin-Resource-Policy: same-site, same-origin` `` ends up as **allowed** below as it will never match anything. Two or more `` `Cross-Origin-Resource-Policy` `` headers will have the same effect.*

4. If *policy* is `` `same-origin` ``, then return **blocked**.
5. If the following are true

   - *request*'s origin's host is same site with *request*'s current URL's host
   - *request*'s origin's scheme is "`https`" or *response*'s HTTPS state is "`none`"

   then return **allowed**.

   Note

   This prevents HTTPS responses with `` `Cross-Origin-Resource-Policy: same-site` `` from being accessed without secure transport.

6. If *policy* is `` `same-site` ``, then return **blocked**.
7. Return **allowed**.

File an issue about the selected text

4. Fetching §

Note

The algorithm below defines fetching. In broad strokes, it takes a request and outputs a response.

That is, it either returns a response if request's synchronous flag is set, or it queues tasks annotated process response, process response end-of-body, and process response done for the response.

To capture uploads, if request's synchronous flag is unset, tasks annotated process request body and process request end-of-body for the request can be queued.

To perform a **fetch** using *request*, run the steps below. An ongoing fetch can be **terminated** with flag *aborted*, which is unset unless otherwise specified.

The user agent may be asked to **suspend** the ongoing fetch. The user agent may either accept or ignore the suspension request. The suspended fetch can be **resumed**. The user agent should ignore the suspension request if the ongoing fetch is updating the response in the HTTP cache for the request.

Note

The user agent does not update the entry in the HTTP cache for a request if request's cache mode is "no-store" or a `` `Cache-Control: no-store` `` header appears in the response. [HTTP-CACHING]

1. Run these steps, but abort when the ongoing fetch is terminated:
    1. If *request*'s window is "`client`", set *request*'s window to *request*'s client, if *request*'s client's global object is a `Window` object, and to "`no-window`" otherwise.
    2. If *request*'s origin is "`client`", set *request*'s origin to *request*'s client's origin.
    3. If *request*'s header list does not contain `` `Accept` ``, then:
        1. Let *value* be `` `*/*` ``.
        2. If *request* is a navigation request, a user agent should set *value* to `` `text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8` ``.
        3. Otherwise, a user agent should set *value* to the first matching statement, if any, switching on *request*'s destination:

            ↪ **"image"**

            `` `image/png,image/svg+xml,image/*;q=0.8,*/*;q=0.5` ``

            ↪ **"style"**

            `` `text/css,*/*;q=0.1` ``
        4. Append `` `Accept` ``/*value* to *request*'s header list.
    4. If *request*'s header list does not contain `` `Accept-Language` ``, user agents should append `` `Accept-Language` ``/an appropriate value to *request*'s header list.
    5. If *request*'s priority is null, then use *request*'s initiator and destination appropriately in setting *request*'s priority to a user-agent-defined object.

        Note

        The user-agent-defined object could encompass stream weight and dependency for HTTP/2, and equivalent information used to prioritize dispatch and processing of HTTP/1 fetches.
    6. If *request* is a subresource request, then:
        1. Let *record* be a new fetch record consisting of *request* and this instance of the fetch algorithm.
        2. Append *record* to *request*'s client's fetch group list of fetch records.
2. If aborted, then:
    1. Let *aborted* be the termination's aborted flag.

File an issue about the selected text

   2. If *aborted* is set, then return an aborted network error.

   3. Return a network error.

3. Return the result of performing a main fetch using *request*.

4.1. Main fetch §

To perform a **main fetch** using *request*, optionally with a *CORS flag* and *recursive flag*, run these steps:

Note

When main fetch is invoked recursively recursive flag *is set.* CORS flag *is a bookkeeping detail for handling redirects.*

1. Let *response* be null.

2. Run these steps, but abort when the ongoing fetch is terminated:

   1. If *request*'s local-URLs-only flag is set and *request*'s current URL is not local, then set *response* to a network error.

   2. Execute Report Content Security Policy violations for *request*. [CSP]

   3. Upgrade *request* to a potentially secure URL, if appropriate. [UPGRADE]

   4. If should fetching *request* be blocked due to a bad port, should fetching *request* be blocked as mixed content, or should fetching *request* be blocked by Content Security Policy returns **blocked**, set *response* to a network error. [MIX] [CSP]

   5. If *request*'s referrer policy is the empty string and *request*'s client is non-null, then set *request*'s referrer policy to *request*'s client's referrer policy. [REFERRER]

   6. If *request*'s referrer policy is the empty string, then set *request*'s referrer policy to "`no-referrer-when-downgrade`".

      Note

      We use "`no-referrer-when-downgrade`" because it is the historical default.

   7. If *request*'s referrer is not "`no-referrer`", set *request*'s referrer to the result of invoking determine *request*'s referrer. [REFERRER]

      Note

      As stated in Referrer Policy, user agents can provide the end user with options to override request*'s referrer to "`no-referrer`" or have it expose less sensitive information.*

   8. If *request*'s current URL's scheme is "`ftp`", *request*'s client's creation URL's scheme is not "`ftp`", and *request*'s reserved client is either null or an environment whose target browsing context is a nested browsing context, then set *response* to a network error.

   9. Set *request*'s current URL's scheme to "`https`" if all of the following conditions are true:

      - *request*'s current URL's scheme is "`http`"
      - *request*'s current URL's host is a domain
      - Matching *request*'s current URL's host per Known HSTS Host Domain Name Matching results in either a superdomain match with an asserted `includeSubDomains` directive or a congruent match (with or without an asserted `includeSubDomains` directive). [HSTS]

3. If aborted, then:

   1. Let *aborted* be the termination's aborted flag.

   2. If *aborted* is set, then return an aborted network error.

   3. Return a network error.

4. If *request*'s synchronous flag is unset and *recursive flag* is unset, run the remaining steps in parallel.

5. If *response* is null, then set *response* to the result of running the steps corresponding to the first matching statement:

   ↪ ***request*'s current URL's origin is same origin with *request*'s origin, *request*'s tainted origin flag is unset, and the *CORS flag* is unset**

   ↪ ***request*'s current URL's scheme is "`data`"**

   ↪ ***request*'s mode is "`navigate`" or "`websocket`"**

      1. Set *request*'s response tainting to "`basic`".

      2. Return the result of performing a scheme fetch using *request*.

File an issue about the selected text

Note

HTML assigns any documents and workers created from URLs whose scheme is "`data`" a unique opaque origin. Service workers can only be created from URLs whose scheme is an HTTP(S) scheme. [HTML] [SW]

↪ ***request*'s mode is "`same-origin`"**

Return a network error.

↪ ***request*'s mode is "`no-cors`"**

1. If *request*'s redirect mode is not "`follow`", then return a network error.
2. Set *request*'s response tainting to "`opaque`".
3. Let *noCorsResponse* be the result of performing a scheme fetch using *request*.
4. If *noCorsResponse* is a filtered response or the CORB check with *request* and *noCorsResponse* returns **allowed**, then return *noCorsResponse*.
5. Return a new response whose status is *noCorsResponse*'s status, HTTPS state is *noCorsResponse*'s HTTPS state, and CSP list is *noCorsResponse*'s CSP list.

⚠Warning!

This is only an effective defense against side channel attacks if* noCorsResponse *is kept isolated from the process that initiated the request.

↪ ***request*'s current URL's scheme is not an HTTP(S) scheme**

Return a network error.

↪ ***request*'s use-CORS-preflight flag is set**

↪ ***request*'s unsafe-request flag is set and either *request*'s method is not a CORS-safelisted method or CORS-unsafe request-header names with *request*'s header list is not empty**

1. Set *request*'s response tainting to "`cors`".
2. Let *corsWithPreflightResponse* be the result of performing an HTTP fetch using *request* with *CORS flag* and *CORS-preflight flag* set.
3. If *corsWithPreflightResponse* is a network error, then clear cache entries using *request*.
4. Return *corsWithPreflightResponse*.

↪ **Otherwise**

1. Set *request*'s response tainting to "`cors`".
2. Return the result of performing an HTTP fetch using *request* with *CORS flag* set.

6. If the *recursive flag* is set, return *response*.
7. If *response* is not a network error and *response* is not a filtered response, then:
    1. If *request*'s response tainting is "`cors`", then:
        1. Let *headerNames* be the result of extracting header list values given `` `Access-Control-Expose-Headers` `` and *response*'s header list.
        2. If *request*'s credentials mode is not "`include`" and *headerNames* contains `` `*` ``, then set *response*'s CORS-exposed header-name list to all unique header names in *response*'s header list.
        3. Otherwise, if *headerNames* is not null or failure, then set *response*'s CORS-exposed header-name list to *headerNames*.

        Note

        One of the headerNames *can still be `` `*` `` at this point, but will only match a header whose name is `` `*` ``.*
    2. Set *response* to the following filtered response with *response* as its internal response, depending on *request*'s response tainting:

        ↪ **"`basic`"**

        basic filtered response

        ↪ **"`cors`"**

        CORS filtered response

        ↪ **"`opaque`"**

File an issue about the selected text   filtered response

8. Let *internalResponse* be *response*, if *response* is a network error, and *response*'s internal response otherwise.

9. If *internalResponse*'s URL list is empty, then set it to a clone of *request*'s URL list.

   Note

   A response's URL list will typically be empty at this point, unless it came from a service worker, in which case it will only be empty if it was created through `new Response()`.

10. Set *internalResponse*'s CSP list. [CSP]

11. If *response* is not a network error and any of the following algorithms returns **blocked**, then set *response* and *internalResponse* to a network error:

    - should *internalResponse* to *request* be blocked as mixed content [MIX]
    - should *internalResponse* to *request* be blocked by Content Security Policy [CSP]
    - should *internalResponse* to *request* be blocked due to its MIME type
    - should *internalResponse* to *request* be blocked due to nosniff

12. If *response*'s type is "`opaque`", *internalResponse*'s status is `206`, *internalResponse*'s range-requested flag is set, and *request*'s header list does not contain `` `Range` ``, then set *response* and *internalResponse* to a network error.

    Note

    Traditionally, APIs accept a ranged response even if a range was not requested. This prevents a partial response from an earlier ranged request being provided to an API that did not make a range request.

    ▶ *Further details*

13. If *response* is not a network error and either *request*'s method is `` `HEAD` `` or `` `CONNECT` ``, or *internalResponse*'s status is a null body status, set *internalResponse*'s body to null and disregard any enqueuing toward it (if any).

    Note

    This standardizes the error handling for servers that violate HTTP.

14. If *response* is not a network error and *request*'s integrity metadata is not the empty string, then:

    1. Wait for *response*'s body.

    2. If *response*'s body's stream has not errored, and *response* does not match *request*'s integrity metadata, set *response* and *internalResponse* to a network error. [SRI]

    Note

    This operates on response *as this algorithm is not supposed to observe* internalResponse. *That would allow an attacker to use hashes as an oracle.*

15. If *request*'s synchronous flag is set, wait for *internalResponse*'s body, and then return *response*.

    Note

    This terminates fetch.

16. If *request*'s current URL's scheme is an HTTP(S) scheme, then:

    1. If *request*'s body is done, queue a fetch-request-done task for *request*.

    2. Otherwise, in parallel, wait for *request*'s body, and then queue a fetch-request-done task for *request*.

17. Queue a fetch task on *request* to process response for *response*.

18. Wait for *internalResponse*'s body.

19. Queue a fetch task on *request* to process response end-of-body for *response*.

20. Wait for either *internalResponse*'s trailer, if any, or for the ongoing fetch to terminate. Note *See section 4.1.2 of [HTTP].*

21. If the ongoing fetch is terminated, then set *internalResponse*'s trailer failed flag.

22. Set *request*'s done flag.

23. Queue a fetch task on *request* to process response done for *response*.

4.2. Scheme fetch §

File an issue about the selected text

To perform a **scheme fetch** using *request*, switch on *request*'s current URL's scheme, and run the associated steps:

↪ **"about"**

If *request*'s current URL's cannot-be-a-base-URL flag is set and path contains a single string "blank", then return a new response whose status message is `OK`, header list consist of a single header whose name is `Content-Type` and value is `text/html;charset=utf-8`, body is the empty byte sequence, and HTTPS state is *request*'s client's HTTPS state if *request*'s client is non-null.

Otherwise, return a network error.

Note

URLs such as "about:config" are handled during navigation and result in a network error in the context of fetching.

↪ **"blob"**

1. Run these steps, but abort when the ongoing fetch is terminated:
    1. Let *blob* be *request*'s current URL's blob URL entry's object.
    2. If *request*'s method is not `GET` or *blob* is not a `Blob` object, then return a network error. [FILEAPI]

       Note

       The `GET` method restriction serves no useful purpose other than being interoperable.
    3. Let *response* be a new response whose status message is `OK`.
    4. Append `Content-Length`/*blob*'s `size` attribute value to *response*'s header list.
    5. Append `Content-Type`/*blob*'s `type` attribute value to *response*'s header list.
    6. Set *response*'s HTTPS state to *request*'s client's HTTPS state if *request*'s client is non-null.
    7. Set *response*'s body to the result of performing the read operation on *blob*.
    8. Return *response*.
2. If aborted, then:
    1. Let *aborted* be the termination's aborted flag.
    2. If *aborted* is set, then return an aborted network error.
    3. Return a network error.

↪ **"data"**

1. Let *dataURLStruct* be the result of running the `data:` URL processor on *request*'s current URL.
2. If *dataURLStruct* is failure, then return a network error.
3. Return a response whose status message is `OK`, header list consist of a single header whose name is `Content-Type` and value is *dataURLStruct*'s MIME type, serialized, body is *dataURLStruct*'s body, and HTTPS state is *request*'s client's HTTPS state if *request*'s client is non-null.

↪ **"file"**

For now, unfortunate as it is, `file` URLs are left as an exercise for the reader.

When in doubt, return a network error.

↪ **"ftp"**

For now, unfortunate as it is, `ftp` URLs are mostly left as an exercise for the reader.

1. Let *body* be the result of the user agent obtaining content from *request*'s current URL from the network via FTP. [RFC959]
2. Let mime be `application/octet-stream`.
3. If *body* is the result of the user agent generating a directory listing page for the result of FTP's LIST command, then set *mime* to `text/ftp-dir`.
4. Return a response whose status message is `OK`, header list consists of a single header whose name is `Content-Type` and whose value is *mime*, body is *body*, and HTTPS state is "none".

When in doubt, return a network error.

↪ **HTTP(S) scheme**

[illegible] rming an HTTP fetch using *request*.

File an issue about the selected text

↪ **Otherwise**

Return a network error.

4.3. HTTP fetch §

To perform an **HTTP fetch** using *request* with an optional *CORS flag* and *CORS-preflight flag*, run these steps:

Note

CORS flag *is still a bookkeeping detail. As is* CORS-preflight flag*; it indicates a CORS-preflight request is needed.*

1. Let *response* be null.
2. Let *actualResponse* be null.
3. If *request*'s service-workers mode is "all", then:
    1. Set *response* to the result of invoking handle fetch for *request*. [HTML] [SW]
    2. If *response* is not null, then:
        1. Transmit body for *request*.
        2. Set *actualResponse* to *response*, if *response* is not a filtered response, and to *response*'s internal response otherwise.
        3. If one of the following is true
            - *response*'s type is "error"
            - *request*'s mode is "same-origin" and *response*'s type is "cors"
            - *request*'s mode is not "no-cors" and *response*'s type is "opaque"
            - *request*'s redirect mode is not "manual" and *response*'s type is "opaqueredirect"
            - *request*'s redirect mode is not "follow" and *response*'s URL list has more than one item.

            then return a network error.
4. If *response* is null, then:
    1. If the *CORS-preflight flag* is set and one of these conditions is true:
        - There is no method cache entry match for *request*'s method using *request*, and either *request*'s method is a not a CORS-safelisted method or *request*'s use-CORS-preflight flag is set.
        - There is at least one item in the CORS-unsafe request-header names with *request*'s header list for which there is no header-name cache entry match using *request*.

        Then:

        1. Let *preflightResponse* be the result of performing a CORS-preflight fetch using *request*.
        2. If *preflightResponse* is a network error, then return *preflightResponse*.

        Note

        This step checks the CORS-preflight cache and if there is no suitable entry it performs a CORS-preflight fetch which, if successful, populates the cache. The purpose of the CORS-preflight fetch is to ensure the fetched resource is familiar with the CORS protocol. The cache is there to minimize the number of CORS-preflight fetches.
    2. If *request*'s redirect mode is "follow", then set *request*'s service-workers mode to "none".

        Note

        Redirects coming from the network (as opposed to from a service worker) are not to be exposed to a service worker.
    3. Set *response* and *actualResponse* to the result of performing an HTTP-network-or-cache fetch using *request* with *CORS flag* if set.
    4. If *CORS flag* is set and a CORS check for *request* and *response* returns failure, then return a network error.

        Note

        As the CORS check is not to be applied to responses whose status is `304` or `407`, or responses from a service worker for that matter, it is applied here.
5. If *actualResponse*'s status is a redirect status, then:
    1. If *actualResponse*'s status is not `303`, *request*'s body is not null, and the connection uses HTTP/2, then user agents may, and are even encouraged to, transmit an `RST_STREAM` frame.

File an issue about the selected text

Note
303 is excluded as certain communities ascribe special status to it.

2. Let *location* be the result of extracting header list values given `` `Location` `` and *actualResponse*'s header list.
3. If *location* is a value, then set *location* to the result of parsing *location* with *actualResponse*'s URL.
4. Set *actualResponse*'s location URL to *location*.
5. Switch on *request*'s redirect mode:

   ↪ **"error"**
   Set *response* to a network error.

   ↪ **"manual"**
   Set *response* to an opaque-redirect filtered response whose internal response is *actualResponse*.

   ↪ **"follow"**
   Set *response* to the result of performing HTTP-redirect fetch using *request* and *response* with *CORS flag* if set.

6. Return *response*. Note *Typically actualResponse's body's stream is still being enqueued to after returning.*

4.4. HTTP-redirect fetch §

Note
This algorithm is used by HTML's navigate algorithm in addition to HTTP fetch above. [HTML]

To perform an **HTTP-redirect fetch** using *request* and *response*, with an optional *CORS flag*, run these steps:

1. Let *actualResponse* be *response*, if *response* is not a filtered response, and *response*'s internal response otherwise.
2. If *actualResponse*'s location URL is null, then return *response*.
3. If *actualResponse*'s location URL is failure, then return a network error.
4. If *actualResponse*'s location URL's scheme is *not* an HTTP(S) scheme, then return a network error.
5. If *request*'s redirect count is twenty, return a network error.
6. Increase *request*'s redirect count by one.
7. If *request*'s mode is "cors", *actualResponse*'s location URL includes credentials, and either *request*'s tainted origin flag is set or *request*'s origin is not same origin with *actualResponse*'s location URL's origin, then return a network error.
8. If *CORS flag* is set and *actualResponse*'s location URL includes credentials, then return a network error.

   Note
   This catches a cross-origin resource redirecting to a same-origin URL.

9. If *actualResponse*'s status is not 303, *request*'s body is non-null, and *request*'s body's source is null, then return a network error.
10. If *actualResponse*'s location URL's origin is not same origin with *request*'s current URL's origin and *request*'s origin is not same origin with *request*'s current URL's origin, then set *request*'s tainted origin flag.
11. If one of the following is true

    - *actualResponse*'s status is 301 or 302 and *request*'s method is `` `POST` ``
    - *actualResponse*'s status is 303 and *request*'s method is not `` `HEAD` ``

    then set *request*'s method to `` `GET` `` and *request*'s body to null.

12. If *request*'s body is non-null, then set *request*'s body to the first return value of safely extracting *request*'s body's source.

    Note
    request's body's source's nullity has already been checked.

13. Append *actualResponse*'s location URL to *request*'s URL list.
14. Invoke set *request*'s referrer policy on redirect on *request* and *actualResponse*. [REFERRER]
15. Return the result of performing a main fetch using *request* with

File an issue about the selected text

- *CORS flag* if set and
- *recursive flag* set if *request*'s redirect mode is not "`manual`".

  Note can only be "`manual`" when invoked directly from HTML's navigate algorithm.

Note

This has to invoke main fetch to get response tainting correct.

4.5. HTTP-network-or-cache fetch §

To perform an **HTTP-network-or-cache fetch** using *request* with an optional *CORS flag* and *authentication-fetch flag*, run these steps:

Note

CORS flag *is still a bookkeeping detail. As is* authentication-fetch flag*.*

Note

Some implementations might support caching of partial content, as per HTTP Range Requests. [HTTP-RANGE] However, this is not widely supported by browser caches.

1. Let *httpRequest* be null.
2. Let *response* be null.
3. Let *storedResponse* be null.
4. Let the *revalidatingFlag* be unset.
5. Run these steps, but abort when the ongoing fetch is terminated:
    1. If *request*'s window is "`no-window`" and *request*'s redirect mode is "`error`", then set *httpRequest* to *request*.
    2. Otherwise:
        1. Set *httpRequest* to a copy of *request* except for its body.
        2. Let *body* be *request*'s body.
        3. Set *httpRequest*'s body to *body*.
        4. If *body* is non-null, then set *request*'s body to a new body whose stream is null and whose source is *body*'s source.

        Note

        request *is copied as* httpRequest *here as we need to be able to add headers to* httpRequest *and read its body without affecting* request*. Namely,* request *can be reused with redirects, authentication, and proxy authentication. We copy rather than clone in order to reduce memory consumption. In case* request*'s body's source is null, redirects and authentication will end up failing the fetch.*
    3. Let *credentials flag* be set if one of
        - *request*'s credentials mode is "`include`"
        - *request*'s credentials mode is "`same-origin`" and *request*'s response tainting is "`basic`"

        is true, and unset otherwise.
    4. Let *contentLengthValue* be null.
    5. If *httpRequest*'s body is null and *httpRequest*'s method is `` `POST` `` or `` `PUT` ``, then set *contentLengthValue* to `` `0` ``.
    6. If *httpRequest*'s body is non-null and *httpRequest*'s body's source is non-null, then set *contentLengthValue* to *httpRequest*'s body's total bytes, serialized and isomorphic encoded.
    7. If *contentLengthValue* is non-null, append `` `Content-Length` ``/*contentLengthValue* to *httpRequest*'s header list.
    8. If *contentLengthValue* is non-null and *httpRequest*'s keepalive flag is set, then:
        1. Let *inflightKeepaliveBytes* be zero.
        2. Let *group* be *httpRequest*'s client's fetch group.
        3. Let *inflightRecords* be the set of fetch records in *group* whose request has its keepalive flag set and done flag unset.
        4. For each *fetchRecord* in *inflightRecords*:
            1. Let *inflightRequest* be *fetchRecord*'s request.

File an issue about the selected text

2. Increment *inflightKeepaliveBytes* by *inflightRequest*'s body's total bytes.

5. If the sum of *contentLengthValue* and *inflightKeepaliveBytes* is greater than 64 kibibytes, then return a network error.

Note

The above limit ensures that requests that are allowed to outlive the environment settings object and contain a body, have a bounded size and are not allowed to stay alive indefinitely.

9. If *httpRequest*'s referrer is a URL, then append `Referer`/*httpRequest*'s referrer, serialized and isomorphic encoded, to *httpRequest*'s header list.

10. If the *CORS flag* is set, *httpRequest*'s method is neither `GET` nor `HEAD`, or *httpRequest*'s mode is "websocket", then append `Origin`/the result of serializing a request origin with *httpRequest*, to *httpRequest*'s header list.

11. If *httpRequest*'s header list does not contain `User-Agent`, then user agents should append `User-Agent`/default `User-Agent` value to *httpRequest*'s header list.

12. If *httpRequest*'s cache mode is "default" and *httpRequest*'s header list contains `If-Modified-Since`, `If-None-Match`, `If-Unmodified-Since`, `If-Match`, or `If-Range`, then set *httpRequest*'s cache mode to "no-store".

13. If *httpRequest*'s cache mode is "no-cache" and *httpRequest*'s header list does not contain `Cache-Control`, then append `Cache-Control`/`max-age=0` to *httpRequest*'s header list.

14. If *httpRequest*'s cache mode is "no-store" or "reload", then:

    1. If *httpRequest*'s header list does not contain `Pragma`, then append `Pragma`/`no-cache` to *httpRequest*'s header list.

    2. If *httpRequest*'s header list does not contain `Cache-Control`, then append `Cache-Control`/`no-cache` to *httpRequest*'s header list.

15. If *httpRequest*'s header list contains `Range`, then append `Accept-Encoding`/`identity` to *httpRequest*'s header list.

Note

This avoids a failure when handling content codings with a part of an encoded response.

Additionally, many servers mistakenly ignore `Range` headers if a non-identity encoding is accepted.

16. Modify *httpRequest*'s header list per HTTP. Do not append a given header if *httpRequest*'s header list contains that header's name.

Note

It would be great if we could make this more normative somehow. At this point headers such as `Accept-Encoding`, `Connection`, `DNT`, and `Host`, are to be appended if necessary.

`Accept`, `Accept-Charset`, and `Accept-Language` must not be included at this point.

Note

`Accept` and `Accept-Language` are already included (unless `fetch()` is used, which does not include the latter by default), and `Accept-Charset` is a waste of bytes. See HTTP header layer division for more details.

17. If *credentials flag* is set, then:

    1. If the user agent is not configured to block cookies for *httpRequest* (see section 7 of [COOKIES]), then:

        1. Let *cookies* be the result of running the "cookie-string" algorithm (see section 5.4 of [COOKIES]) with the user agent's cookie store and *httpRequest*'s current URL.

        2. If *cookies* is not the empty string, append `Cookie`/*cookies* to *httpRequest*'s header list.

    2. If *httpRequest*'s header list does not contain `Authorization`, then:

        1. Let *authorizationValue* be null.

        2. If there's an authentication entry for *httpRequest* and either *httpRequest*'s use-URL-credentials flag is unset or *httpRequest*'s current URL does not include credentials, then set *authorizationValue* to authentication entry.

        3. Otherwise, if *httpRequest*'s current URL does include credentials and *authentication-fetch flag* is set, set *authorizationValue* to *httpRequest*'s current URL, converted to an `Authorization` value.

        4. If *authorizationValue* is non-null, then append `Authorization`/*authorizationValue* to *httpRequest*'s header list.

18. If there's a proxy-authentication entry, use it as appropriate.

File an issue about the selected text

Note

This intentionally does not depend on httpRequest*'s [credentials mode](#).*

19. If *httpRequest*'s [cache mode](#) is neither "`no-store`" nor "`reload`", then:

    1. Set *storedResponse* to the result of selecting a response from the HTTP cache, possibly needing validation, as per the "[Constructing Responses from Caches](#)" chapter of *HTTP Caching* [HTTP-CACHING], if any.

       Note

       As mandated by HTTP, this still takes the `` `Vary` `` [header](#) into account.

    2. If *storedResponse* is non-null, then:

       1. If *storedResponse* requires validation (i.e., it is not fresh), then set the *revalidatingFlag*.

       2. If the *revalidatingFlag* is set and *httpRequest*'s [cache mode](#) is neither "`force-cache`" nor "`only-if-cached`", then:

          1. If *storedResponse*'s [header list](#) [contains](#) `` `ETag` ``, then [append](#) `` `If-None-Match` `` with its value to *httpRequest*'s [header list](#).

          2. If *storedResponse*'s [header list](#) [contains](#) `` `Last-Modified` ``, then [append](#) `` `If-Modified-Since` `` with its value to *httpRequest*'s [header list](#).

          Note

          See also the "[Sending a Validation Request](#)" chapter of HTTP Caching [HTTP-CACHING].

       3. Otherwise, set *response* to *storedResponse* and set *response*'s [cache state](#) to "`local`".

6. [If aborted](#), then:

   1. Let *aborted* be the termination's aborted flag.

   2. If *aborted* is set, then return an [aborted network error](#).

   3. Return a [network error](#).

7. If *response* is null, then:

   1. If *httpRequest*'s [cache mode](#) is "`only-if-cached`", then return a [network error](#).

   2. Let *forwardResponse* be the result of making an [HTTP-network fetch](#) using *httpRequest* with *credentials flag* if set.

   3. If *httpRequest*'s [method](#) is [unsafe](#) and *forwardResponse*'s [status](#) is in the range `200` to `399`, inclusive, invalidate appropriate stored responses in the HTTP cache, as per the "[Invalidation](#)" chapter of *HTTP Caching*, and set *storedResponse* to null. [HTTP-CACHING]

   4. If the *revalidatingFlag* is set and *forwardResponse*'s [status](#) is `304`, then:

      1. Update *storedResponse*'s [header list](#) using *forwardResponse*'s [header list](#), as per the "[Freshening Stored Responses upon Validation](#)" chapter of *HTTP Caching*. [HTTP-CACHING]

         Note

         This updates the stored response in cache as well.

      2. Set *response* to *storedResponse*.

   5. If *response* is null, then:

      1. Set *response* to *forwardResponse*.

      2. Store *httpRequest* and *forwardResponse* in the HTTP cache, as per the "[Storing Responses in Caches](#)" chapter of *HTTP Caching*. [HTTP-CACHING]

         Note

         If forwardResponse *is a [network error](#), this effectively caches the network error, which is sometimes known as "negative caching".*

8. If *httpRequest*'s [header list](#) [contains](#) `` `Range` ``, then set *response*'s [range-requested flag](#).

9. If the *CORS flag* is unset and the [cross-origin resource policy check](#) with *request* and *response* returns **blocked**, then return a [network error](#).

10. If *response*'s [status](#) is `401`, *CORS flag* is unset, *credentials flag* is set, and *request*'s [window](#) is an [environment settings object](#), then:

    1. Needs testing: multiple `` `WWW-Authenticate` `` headers, missing, parsing issues.

File an issue about the selected text

   2. If *request*'s body is non-null, then:
      1. If *request*'s body's source is null, then return a network error.
      2. Set *request*'s body to the first return value of safely extracting *request*'s body's source.
   3. If *request*'s use-URL-credentials flag is unset or *authentication-fetch flag* is set, then:
      1. If the ongoing fetch is terminated, then:
         1. Let *aborted* be the termination's aborted flag.
         2. If *aborted* is set, then return an aborted network error.
         3. Return a network error.
      2. Let *username* and *password* be the result of prompting the end user for a username and password, respectively, in *request*'s window.
      3. Set the username given *request*'s current URL and *username*.
      4. Set the password given *request*'s current URL and *password*.
   4. Set *response* to the result of performing an HTTP-network-or-cache fetch using *request* with *authentication-fetch flag* set.

11. If *response*'s status is `407`, then:
    1. If *request*'s window is "`no-window`", then return a network error.
    2. Needs testing: multiple `` `Proxy-Authenticate` `` headers, missing, parsing issues.
    3. If the ongoing fetch is terminated, then:
       1. Let *aborted* be the termination's aborted flag.
       2. If *aborted* is set, then return an aborted network error.
       3. Return a network error.
    4. Prompt the end user as appropriate in *request*'s window and store the result as a proxy-authentication entry. [HTTP-AUTH]

       Note

       Remaining details surrounding proxy authentication are defined by HTTP.
    5. Set *response* to the result of performing an HTTP-network-or-cache fetch using *request* with *CORS flag* if set.
12. If *authentication-fetch flag* is set, then create an authentication entry for *request* and the given realm.
13. Return *response*. Note *Typically response's body's stream is still being enqueued to after returning.*

4.6. HTTP-network fetch §

To perform an **HTTP-network fetch** using *request* with an optional *credentials flag*, run these steps:

1. Let *credentials* be true if *credentials flag* is set, and false otherwise.
2. Let *response* be null.
3. Switch on *request*'s mode:

   "`websocket`"

   Let *connection* be the result of obtaining a WebSocket connection, given *request*'s current URL.

   Otherwise

   Let *connection* be the result of obtaining a connection, given *request*'s current URL's origin and *credentials*.
4. Run these steps, but abort when the ongoing fetch is terminated:
   1. If *connection* is failure, return a network error.

File an issue about the selected text

2. If *connection* is not an HTTP/2 connection, *request*'s body is non-null, and *request*'s body's source is null, then append `` `Transfer-Encoding` ``/`` `chunked` `` to *request*'s header list.

3. Set *response* to the result of making an HTTP request over *connection* using *request* with the following caveats:

    - Follow the relevant requirements from HTTP. [HTTP] [HTTP-SEMANTICS] [HTTP-COND] [HTTP-CACHING] [HTTP-AUTH]

    - Wait until all the headers are transmitted.

    - Any responses whose status is in the range `100` to `199`, inclusive, and is not `101`, are to be ignored.

      Note

      These kind of responses are eventually followed by a "final" response.

   Note

   The exact layering between Fetch and HTTP still needs to be sorted through and therefore response *represents both a response and an HTTP response here.*

   If *request*'s header list contains `` `Transfer-Encoding` ``/`` `chunked` `` and *response* is transferred via HTTP/1.0 or older, then return a network error.

   If the HTTP request results in a TLS client certificate dialog, then:

    1. If *request*'s window is an environment settings object, make the dialog available in *request*'s window.

    2. Otherwise, return a network error.

   If *response* was retrieved over HTTPS, set its HTTPS state to either "`deprecated`" or "`modern`". [TLS]

   Note

   The exact determination here is up to user agents for the time being. User agents are strongly encouraged to only succeed HTTPS connections with strong security properties and return network errors otherwise. Using the "`deprecated`" state value ought to be a temporary and last resort kind of option.

   Transmit body for *request*.

5. If aborted, then:

    1. Let *aborted* be the termination's aborted flag.

    2. If *connection* uses HTTP/2, then transmit an `RST_STREAM` frame.

    3. If *aborted* is set, then return an aborted network error.

    4. Return a network error.

6. Let *highWaterMark* be a non-negative, non-NaN number, chosen by the user agent.

7. Let *sizeAlgorithm* be an algorithm that accepts a chunk object and returns a non-negative, non-NaN, non-infinite number, chosen by the user agent.

8. Let *pull* be an action that resumes the ongoing fetch if it is suspended.

9. Let *cancel* be an action that terminates the ongoing fetch with the aborted flag set.

10. Let *stream* be the result of constructing a `ReadableStream` object with *highWaterMark*, *sizeAlgorithm*, *pull*, and *cancel*.

    Note

    This construction operation will not throw an exception.

11. Run these steps, but abort when the ongoing fetch is terminated:

    1. Set *response*'s body to a new body whose stream is *stream*.

    2. If *response* has a payload body length, then set *response*'s body's total bytes to that payload body length.

    3. If *response* is not a network error and *request*'s cache mode is not "`no-store`", update *response* in the HTTP cache for *request*.

    4. If *credentials flag* is set and the user agent is not configured to block cookies for *request* (see section 7 of [COOKIES]), then run the "set-cookie-string" parsing algorithm (see section 5.2 of [COOKIES]) on the value of each *header* whose name is a byte-case-insensitive match for `` `Set-Cookie` `` in *response*'s header list, if any, and *request*'s current URL.

       Note

       This is a fingerprinting vector.

File an issue about the selected text

12. If aborted, then:
    1. Let *aborted* be the termination's aborted flag.
    2. If *aborted* is set, then set *response*'s aborted flag.
    3. Return *response*.
13. Run these steps in parallel:
    1. Run these steps, but abort when the ongoing fetch is terminated:
        1. While true:
            1. If one or more bytes have been transmitted from *response*'s message body, then:
                1. Let *bytes* be the transmitted bytes.
                2. Increase *response*'s body's transmitted bytes with *bytes*' length.
                3. Let *codings* be the result of extracting header list values given `` `Content-Encoding` `` and *response*'s header list.
                4. Set *bytes* to the result of handling content codings given *codings* and *bytes*.

                   Note

                   This makes the `` `Content-Length` `` header unreliable to the extent that it was reliable to begin with.
                5. If *bytes* is failure, then terminate the ongoing fetch.
                6. Enqueue a `Uint8Array` object wrapping an `ArrayBuffer` containing *bytes* to *stream*. If that threw an exception, terminate the ongoing fetch, and error *stream* with that exception.
                7. If *stream* doesn't need more data and *request*'s synchronous flag is unset, ask the user agent to suspend the ongoing fetch.
            2. Otherwise, if the bytes transmission for *response*'s message body is done normally and *stream* is readable, then close *stream* and abort these in-parallel steps.
    2. If aborted, then:
        1. Let *aborted* be the termination's aborted flag.
        2. If *aborted* is set, then:
            1. Set *response*'s aborted flag.
            2. If *stream* is readable, error *stream* with an "`AbortError`" `DOMException`.
        3. Otherwise, if *stream* is readable, error *stream* with a `TypeError`.
        4. If *connection* uses HTTP/2, then transmit an `RST_STREAM` frame.
        5. Otherwise, the user agent should close *connection* unless it would be bad for performance to do so.

           Note

           For instance, the user agent could keep the connection open if it knows there's only a few bytes of transfer remaining on a reusable connection. In this case it could be worse to close the connection and go through the handshake process again for the next fetch.

    Note

    These are run in parallel as at this point it is unclear whether response*'s body is relevant (*response *might be a redirect).*
14. Return *response*. Note *Typically* response*'s body's stream is still being enqueued to after returning.*

4.7. CORS-preflight fetch §

Note

This is effectively the user agent implementation of the check to see if the CORS protocol is understood. The so-called CORS-preflight request. If successful it populates the CORS-preflight cache to minimize the number of these fetches.

ch using *request*, run these steps:

File an issue about the selected text

1. Let *preflight* be a new request whose method is `` `OPTIONS` ``, URL is *request*'s current URL, initiator is *request*'s initiator, destination is *request*'s destination, origin is *request*'s origin, referrer is *request*'s referrer, referrer policy is *request*'s referrer policy, and tainted origin flag is *request*'s tainted origin flag.

   Note

   The service-workers mode of preflight *does not matter as this algorithm uses HTTP-network-or-cache fetch rather than HTTP fetch.*

2. Append `` `Access-Control-Request-Method` `` to *request*'s method in *preflight*'s header list.

3. Let *headers* be the CORS-unsafe request-header names with *request*'s header list.

4. If *headers* is not empty, then:

   1. Let *value* be the items in *headers* separated from each other by `` `,` ``.

   2. Append `` `Access-Control-Request-Headers` `` to *value* in *preflight*'s header list.

   Note

   This intentionally does not use combine, as 0x20 following 0x2C is not the way this was implemented, for better or worse.

5. Let *response* be the result of performing an HTTP-network-or-cache fetch using *preflight* with the *CORS flag* set.

6. If a CORS check for *request* and *response* returns success and *response*'s status is an ok status, then:

   Note

   The CORS check is done on request *rather than* preflight *to ensure the correct credentials mode is used.*

   1. Let *methods* be the result of extracting header list values given `` `Access-Control-Allow-Methods` `` and *response*'s header list.

   2. Let *headerNames* be the result of extracting header list values given `` `Access-Control-Allow-Headers` `` and *response*'s header list.

   3. If either *methods* or *headerNames* is failure, return a network error.

   4. If *methods* is null and *request*'s use-CORS-preflight flag is set, then set *methods* to a new list containing *request*'s method.

      Note

      This ensures that a CORS-preflight fetch that happened due to request*'s use-CORS-preflight flag being set is cached.*

   5. If *request*'s method is not in *methods*, *request*'s method is not a CORS-safelisted method, and *request*'s credentials mode is "`include`" or *methods* does not contain `` `*` ``, then return a network error.

   6. If one of *request*'s header list's names is a CORS non-wildcard request-header name and is not a byte-case-insensitive match for an item in *headerNames*, then return a network error.

   7. For each *unsafeName* in the CORS-unsafe request-header names with *request*'s header list, if *unsafeName* is not a byte-case-insensitive match for an item in *headerNames* and *request*'s credentials mode is "`include`" or *headerNames* does not contain `` `*` ``, return a network error.

   8. Let *max-age* be the result of extracting header list values given `` `Access-Control-Max-Age` `` and *response*'s header list.

   9. If *max-age* is failure or null, then set *max-age* to zero.

   10. If *max-age* is greater than an imposed limit on max-age, then set *max-age* to the imposed limit.

   11. If the user agent does not provide for a cache, then return *response*.

   12. For each *method* in *methods* for which there is a method cache entry match using *request*, set matching entry's max-age to *max-age*.

   13. For each *method* in *methods* for which there is no method cache entry match using *request*, create a new cache entry with *request*, *max-age*, *method*, and null.

   14. For each *headerName* in *headerNames* for which there is a header-name cache entry match using *request*, set matching entry's max-age to *max-age*.

   15. For each *headerName* in *headerNames* for which there is no header-name cache entry match using *request*, create a new cache entry with *request*, *max-age*, null, and *headerName*.

   16. Return *response*.

7. Otherwise, return a network error.

File an issue about the selected text

4.8. CORS-preflight cache §

A user agent has an associated CORS-preflight cache. A **CORS-preflight cache** is a list of cache entries.

A **cache entry** consists of:

- **serialized origin** (a byte sequence)
- **URL** (a URL)
- **max-age** (a number of seconds)
- **credentials** (a boolean)
- **method** (null, `` `*` ``, or a method)
- **header name** (null, `` `*` ``, or a header name)

Cache entries must be removed after the seconds specified in their max-age field have passed since storing the entry. Cache entries may be removed before that moment arrives.

To **create a new cache entry**, given *request*, *max-age*, *method*, and *headerName*, run these steps:

1. Let *entry* be a cache entry, initialized as follows:

   serialized origin

   The result of serializing a request origin with *request*

   URL

   request's current URL

   max-age

   max-age

   credentials

   True if *request*'s credentials mode is "`include`", and false otherwise

   method

   method

   header name

   headerName

2. Append *entry* to the user agent's CORS-preflight cache.

To **clear cache entries**, given a *request*, remove any cache entries in the user agent's CORS-preflight cache whose serialized origin is the result of serializing a request origin with *request* and whose URL is *request*'s current URL.

There is a **cache entry match** for a cache entry *entry* with *request* if *entry*'s serialized origin is the result of serializing a request origin with *request*, *entry*'s URL is *request*'s current URL, and one of

- *entry*'s credentials is true
- *entry*'s credentials is false and *request*'s credentials mode is not "`include`".

is true.

There is a **method cache entry match** for *method* using *request* when there is a cache entry in the user agent's CORS-preflight cache for which there is a cache entry match with *request* and its method is *method* or `` `*` ``.

There is a **header-name cache entry match** for *headerName* using *request* when there is a cache entry in the user agent's CORS-preflight cache for which there is a cache entry match with *request* and one of

- its header name is a byte-case-insensitive match for *headerName*
- its header name is `` `*` `` and *headerName* is not a CORS non-wildcard request-header name

is true.

4.9. CORS check §

To perform a **CORS check** for a *request* and *response*, run these steps:

1. Let *origin* be the result of getting `` `Access-Control-Allow-Origin` `` from *response*'s header list.

2. If *origin* is null, then return failure.

File an issue about the selected text

Note

Null is not `null`.

3. If *request*'s credentials mode is not "`include`" and *origin* is `` `*` ``, then return success.
4. If the result of serializing a request origin with *request* is not *origin*, then return failure.
5. If *request*'s credentials mode is not "`include`", then return success.
6. Let *credentials* be the result of getting `` `Access-Control-Allow-Credentials` `` from *response*'s header list.
7. If *credentials* is `` `true` ``, then return success.
8. Return failure.

File an issue about the selected text

5. Fetch API §

The `fetch()` method is relatively low-level API for fetching resources. It covers slightly more ground than `XMLHttpRequest`, although it is currently lacking when it comes to request progression (not response progression).

Example

The `fetch()` method makes it quite straightforward to fetch a resource and extract its contents as a `Blob`:

```
fetch("/music/pk/altes-kamuffel.flac")
  .then(res => res.blob()).then(playBlob)
```

If you just care to log a particular response header:

```
fetch("/", {method:"HEAD"})
  .then(res => log(res.headers.get("strict-transport-security")))
```

If you want to check a particular response header and then process the response of a cross-origin resources:

```
fetch("https://pk.example/berlin-calling.json", {mode:"cors"})
  .then(res => {
    if(res.headers.get("content-type") &&
       res.headers.get("content-type").toLowerCase().indexOf("application/json") >= 0) {
      return res.json()
    } else {
      throw new TypeError()
    }
  }).then(processJSON)
```

If you want to work with URL query parameters:

```
var url = new URL("https://geo.example.org/api"),
    params = {lat:35.696233, long:139.570431}
Object.keys(params).forEach(key => url.searchParams.append(key, params[key]))
fetch(url).then(/* … */)
```

If you want to receive the body data progressively:

```
function consume(reader) {
  var total = 0
  return pump()
  function pump() {
    return reader.read().then(({done, value}) => {
      if (done) {
        return
      }
      total += value.byteLength
      log(`received ${value.byteLength} bytes (${total} bytes in total)`)
      return pump()
    })
  }
}

fetch("/music/pk/altes-kamuffel.flac")
  .then(res => consume(res.body.getReader()))
  .then(() => log("consumed the entire body without keeping the whole thing in memory!"))
  .catch(e => log("something went wrong: " + e))
```

File an issue about the selected text

5.1. Headers class §

```
typedef (sequence<sequence<ByteString>> or record<ByteString, ByteString>) HeadersInit;

[Constructor(optional HeadersInit init),
 Exposed=(Window,Worker)]
interface Headers {
  void append(ByteString name, ByteString value);
  void delete(ByteString name);
  ByteString? get(ByteString name);
  boolean has(ByteString name);
  void set(ByteString name, ByteString value);
  iterable<ByteString, ByteString>;
};
```

Note

Unlike a header list, a `Headers` object cannot represent more than one `Set-Cookie` header. In a way this is problematic as unlike all other headers `Set-Cookie` headers cannot be combined, but since `Set-Cookie` headers are not exposed to client-side JavaScript this is deemed an acceptable compromise. Implementations could chose the more efficient `Headers` object representation even for a header list, as long as they also support an associated data structure for `Set-Cookie` headers.

Example

A `Headers` object can be initialized with various JavaScript data structures:

```
var meta = { "Content-Type": "text/xml", "Breaking-Bad": "<3" }
new Headers(meta)

// The above is equivalent to
var meta = [
  [ "Content-Type", "text/xml" ],
  [ "Breaking-Bad", "<3" ]
]
new Headers(meta)
```

A `Headers` object has an associated **header list** (a header list), which is initially empty. Note *This can be a pointer to the header list of something else, e.g., of a request as demonstrated by `Request` objects.*

A `Headers` object also has an associated **guard**, which is "`immutable`", "`request`", "`request-no-cors`", "`response`" or "`none`".

To **append** a name/value *name*/*value* pair to a `Headers` object (*headers*), run these steps:

1. Normalize *value*.
2. If *name* is not a name or *value* is not a value, then throw a `TypeError`.
3. If *headers*'s guard is "`immutable`", then throw a `TypeError`.
4. Otherwise, if *headers*'s guard is "`request`" and *name* is a forbidden header name, return.
5. Otherwise, if *headers*'s guard is "`request-no-cors`":
    1. Let *temporaryValue* be the result of getting *name* from *headers*'s header list.
    2. If *temporaryValue* is null, then set *temporaryValue* to *value*.
    3. Otherwise, set *temporaryValue* to *temporaryValue*, followed by 0x2C 0x20, followed by *value*.
    4. If *name*/*temporaryValue* is not a no-CORS-safelisted request-header, then return.
6. Otherwise, if *headers*'s guard is "`response`" and *name* is a forbidden response-header name, return.
7. Append *name*/*value* to *headers*'s header list.
8. If *headers*'s guard is "`request-no-cors`", then remove privileged no-CORS request headers from *headers*.

To **fill** a `Headers` object (*headers*) with a given object (*object*), run these steps:

1. If *object* is a sequence, then for each *header* in *object*:
    1. If *header* does not contain exactly two items, then throw a `TypeError`.

File an issue about the selected text

    2. Append *header*'s first item/*header*'s second item to *headers*.

2. Otherwise, *object* is a record, then for each *key* → *value* in *object*, append *key*/*value* to *headers*.

To **remove privileged no-CORS request headers** from a `Headers` object (*headers*), run these steps:

1. For each *headerName* of privileged no-CORS request-header names:

    1. Delete *headerName* from *headers*'s header list.

Note

This is called when headers are modified by unprivileged code.

The **`Headers(init)`** constructor, when invoked, must run these steps:

1. Let *headers* be a new `Headers` object whose guard is "`none`".
2. If *init* is given, then fill *headers* with *init*.
3. Return *headers*.

The **`append(name, value)`** method, when invoked, must append *name*/*value* to the context object.

The **`delete(name)`** method, when invoked, must run these steps:

1. If *name* is not a name, then throw a `TypeError`.
2. If the context object's guard is "`immutable`", then throw a `TypeError`.
3. Otherwise, if the context object's guard is "`request`" and *name* is a forbidden header name, return.
4. Otherwise, if the context object's guard is "`request-no-cors`", *name* is not a no-CORS-safelisted request-header name, and *name* is not a privileged no-CORS request-header name, return.
5. Otherwise, if the context object's guard is "`response`" and *name* is a forbidden response-header name, return.
6. If the context object's header list does not contain *name*, then return.
7. Delete *name* from the context object's header list.
8. If the context object's guard is "`request-no-cors`", then remove privileged no-CORS request headers from the context object.

The **`get(name)`** method, when invoked, must run these steps:

1. If *name* is not a name, then throw a `TypeError`.
2. Return the result of getting *name* from the context object's header list.

The **`has(name)`** method, when invoked, must run these steps:

1. If *name* is not a name, then throw a `TypeError`.
2. Return true if the context object's header list contains *name*, and false otherwise.

The **`set(name, value)`** method, when invoked, must run these steps:

1. Normalize *value*.
2. If *name* is not a name or *value* is not a value, then throw a `TypeError`.
3. If the context object's guard is "`immutable`", then throw a `TypeError`.
4. Otherwise, if the context object's guard is "`request`" and *name* is a forbidden header name, return.
5. Otherwise, if the context object's guard is "`request-no-cors`" and *name*/*value* is not a no-CORS-safelisted request-header, return.
6. Otherwise, if the context object's guard is "`response`" and *name* is a forbidden response-header name, return.
7. Set *name*/*value* in the context object's header list.
8. If the context object's guard is "`request-no-cors`", then remove privileged no-CORS request headers from the context object.

The value pairs to iterate over are the return value of running sort and combine with the context object's header list.

File an issue about the selected text

5.2. Body mixin §

```
IDL typedef (Blob or BufferSource or FormData or URLSearchParams or ReadableStream or USVString)
BodyInit;
```

To **safely extract** a body and a `Content-Type` value from *object*, run these steps:

1. If *object* is a `ReadableStream` object, then:
    1. Assert: *object* is neither disturbed nor locked.
2. Return the results of extracting *object*.

Note

The safely extract operation is a subset of the extract operation that is guaranteed to not throw an exception.

To **extract** a body and a `Content-Type` value from *object*, with an optional *keepalive flag*, run these steps:

1. Let *stream* be the result of constructing a `ReadableStream` object.
2. Let *Content-Type* be null.
3. Let *action* be null.
4. Let *source* be null.
5. Switch on *object*'s type:

↪ **`Blob`**

Set *action* to an action that reads *object*.

If *object*'s `type` attribute is not the empty byte sequence, set *Content-Type* to its value.

Set *source* to *object*.

↪ **`BufferSource`**

Enqueue a `Uint8Array` object wrapping an `ArrayBuffer` containing a copy of the bytes held by *object* to *stream* and close *stream*. If that threw an exception, error *stream* with that exception.

Set *source* to *object*.

↪ **`FormData`**

Set *action* to an action that runs the `multipart/form-data` encoding algorithm, with *object* as *form data set* and with UTF-8 as the explicit character encoding.

Set *Content-Type* to `multipart/form-data; boundary=`, followed by the `multipart/form-data` boundary string generated by the `multipart/form-data` encoding algorithm.

Set *source* to *object*.

↪ **`URLSearchParams`**

Set *action* to an action that runs the `application/x-www-form-urlencoded` serializer with *object*'s list.

Set *Content-Type* to `application/x-www-form-urlencoded;charset=UTF-8`.

Set *source* to *object*.

↪ **`USVString`**

Set *action* to an action that runs UTF-8 encode on *object*.

Set *Content-Type* to `text/plain;charset=UTF-8`.

Set *source* to *object*.

↪ **`ReadableStream`**

If the *keepalive flag* is set, then throw a `TypeError`.

If *object* is disturbed or locked, then throw a `TypeError`.

Set *stream* to *object*.

File an issue about the selected text ıaction in parallel:

1. Whenever one or more bytes are available, let *bytes* be the bytes and enqueue a `Uint8Array` object wrapping an `ArrayBuffer` containing *bytes* to *stream*. If creating the `ArrayBuffer` threw an exception, error *stream* with that exception and cancel running *action*.

2. When running *action* is done, close *stream*.

7. Let *body* be a body whose stream is *stream* and whose source is *source*.

8. Return *body* and *Content-Type*.

IDL
```
interface mixin Body {
  readonly attribute ReadableStream? body;
  readonly attribute boolean bodyUsed;
  [NewObject] Promise<ArrayBuffer> arrayBuffer();
  [NewObject] Promise<Blob> blob();
  [NewObject] Promise<FormData> formData();
  [NewObject] Promise<any> json();
  [NewObject] Promise<USVString> text();
};
```

Note

Formats you would not want a network layer to be dependent upon, such as HTML, will likely not be exposed here. Rather, an HTML parser API might accept a stream in due course.

Objects implementing the Body mixin gain an associated **body** (null or a body) and a **MIME type** (failure or a MIME type).

An object implementing the Body mixin is said to be **disturbed** if body is non-null and its stream is disturbed.

An object implementing the Body mixin is said to be **locked** if body is non-null and its stream is locked.

The **body** attribute's getter must return null if body is null and body's stream otherwise.

The **bodyUsed** attribute's getter must return true if disturbed, and false otherwise.

Objects implementing the Body mixin also have an associated **package data** algorithm, given *bytes*, a *type* and a *mimeType*, switches on *type*, and runs the associated steps:

↪ ***ArrayBuffer***

Return a new `ArrayBuffer` whose contents are *bytes*.

Note

Allocating an `ArrayBuffer` can throw a `RangeError`.

↪ ***Blob***

Return a `Blob` whose contents are *bytes* and `type` attribute is *mimeType*.

↪ ***FormData***

If *mimeType*'s essence is "`multipart/form-data`", then:

1. Parse *bytes*, using the value of the `` `boundary` `` parameter from *mimeType*, per the rules set forth in *Returning Values from Forms: multipart/form-data*. [RFC7578]

   Each part whose `` `Content-Disposition` `` header contains a `` `filename` `` parameter must be parsed into an entry whose value is a `File` object whose contents are the contents of the part. The `name` attribute of the `File` object must have the value of the `` `filename` `` parameter of the part. The `type` attribute of the `File` object must have the value of the `` `Content-Type` `` header of the part if the part has such header, and `` `text/plain` `` (the default defined by [RFC7578] section 4.4) otherwise.

   Each part whose `` `Content-Disposition` `` header does not contain a `` `filename` `` parameter must be parsed into an entry whose value is the UTF-8 decoded content of the part. Note *This is done regardless of the presence or the value of a `` `Content-Type` `` header and regardless of the presence or the value of a `` `charset` `` parameter.*

   Note

   A part whose `` `Content-Disposition` `` header contains a `` `name` `` parameter whose value is `` `_charset_` `` is parsed like any other part. It does not change the encoding.

2. If that fails for some reason, then throw a `TypeError`.

3. Return a new `FormData` object, appending each entry, resulting from the parsing operation, to entries.

File an issue about the selected text

The above is a rough approximation of what is needed for `multipart/form-data`, a more detailed parsing specification is to be written. Volunteers welcome.

Otherwise, if *mimeType*'s essence is "`application/x-www-form-urlencoded`", then:

1. Let *entries* be the result of parsing *bytes*.
2. If *entries* is failure, then throw a `TypeError`.
3. Return a new `FormData` object whose entries are *entries*.

Otherwise, throw a `TypeError`.

↪ ***JSON***

Return the result of running parse JSON from bytes on *bytes*.

↪ ***text***

Return the result of running UTF-8 decode on *bytes*.

Objects implementing the Body mixin also have an associated **consume body** algorithm, given a *type*, runs these steps:

1. If this object is disturbed or locked, return a new promise rejected with a `TypeError`.
2. Let *stream* be body's stream if body is non-null, or an empty `ReadableStream` object otherwise.
3. Let *reader* be the result of getting a reader from *stream*. If that threw an exception, return a new promise rejected with that exception.
4. Let *promise* be the result of reading all bytes from *stream* with *reader*.
5. Return the result of transforming *promise* by a fulfillment handler that returns the result of the package data algorithm with its first argument, *type* and this object's MIME type.

The **`arrayBuffer()`** method, when invoked, must return the result of running consume body with *ArrayBuffer*.

The **`blob()`** method, when invoked, must return the result of running consume body with *Blob*.

The **`formData()`** method, when invoked, must return the result of running consume body with *FormData*.

The **`json()`** method, when invoked, must return the result of running consume body with *JSON*.

The **`text()`** method, when invoked, must return the result of running consume body with *text*.

5.3. Request class §

IDL
```
typedef (Request or USVString) RequestInfo;

[Constructor(RequestInfo input, optional RequestInit init),
 Exposed=(Window,Worker)]
interface Request {
  readonly attribute ByteString method;
  readonly attribute USVString url;
  [SameObject] readonly attribute Headers headers;

  readonly attribute RequestDestination destination;
  readonly attribute USVString referrer;
  readonly attribute ReferrerPolicy referrerPolicy;
  readonly attribute RequestMode mode;
  readonly attribute RequestCredentials credentials;
  readonly attribute RequestCache cache;
  readonly attribute RequestRedirect redirect;
  readonly attribute DOMString integrity;
  readonly attribute boolean keepalive;
  readonly attribute boolean isReloadNavigation;
  readonly attribute boolean isHistoryNavigation;
  readonly attribute AbortSignal signal;

  [NewObject] Request clone();
```

File an issue about the selected text

```
Request includes Body;

dictionary RequestInit {
  ByteString method;
  HeadersInit headers;
  BodyInit? body;
  USVString referrer;
  ReferrerPolicy referrerPolicy;
  RequestMode mode;
  RequestCredentials credentials;
  RequestCache cache;
  RequestRedirect redirect;
  DOMString integrity;
  boolean keepalive;
  AbortSignal? signal;
  any window; // can only be set to null
};

enum RequestDestination { "", "audio", "audioworklet", "document", "embed", "font", "image",
"manifest", "object", "paintworklet", "report", "script", "sharedworker", "style",  "track", "video",
"worker", "xslt" };
enum RequestMode { "navigate", "same-origin", "no-cors", "cors" };
enum RequestCredentials { "omit", "same-origin", "include" };
enum RequestCache { "default", "no-store", "reload", "no-cache", "force-cache", "only-if-cached" };
enum RequestRedirect { "follow", "error", "manual" };
```

Note

"serviceworker" is omitted from RequestDestination as it cannot be observed from JavaScript. Implementations will still need to support it as a destination. "websocket" is omitted from RequestMode as it cannot be used nor observed from JavaScript.

A Request object has an associated **request** (a request).

A Request object also has an associated **headers** (null or a Headers object), initially null.

A Request object has an associated **signal** (an AbortSignal object), initially a new AbortSignal object.

A Request object's body is its request's body.

For web developers (non-normative)

***request* = new Request(*input* [, *init*])**

Returns a new *request* whose url property is *input* if *input* is a string, and *input*'s url if *input* is a Request object.

The *init* argument is an object whose properties can be set as follows:

method

A string to set *request*'s method.

headers

A Headers object, an object literal, or an array of two-item arrays to set *request*'s headers.

body

A BodyInit object or null to set *request*'s body.

referrer

A string whose value is a same-origin URL, "about:client", or the empty string, to set *request*'s referrer.

referrerPolicy

A referrer policy to set *request*'s referrerPolicy.

mode

A string to indicate whether the request will use CORS, or will be restricted to same-origin URLs. Sets *request*'s mode.

credentials

A string indicating whether credentials will be sent with the request always, never, or only when sent to a same-origin URL. Sets *request*'s credentials.

cache

A string indicating how the request will interact with the browser's cache to set *request*'s cache.

File an issue about the selected text

`redirect`

A string indicating whether *request* follows redirects, results in an error upon encountering a redirect, or returns the redirect (in an opaque fashion). Sets *request*'s `redirect`.

`integrity`

A cryptographic hash of the resource to be fetched by *request*. Sets *request*'s `integrity`.

`keepalive`

A boolean to set *request*'s `keepalive`.

`signal`

An `AbortSignal` to set *request*'s `signal`.

`window`

Can only be null. Used to disassociate *request* from any `Window`.

`request` **`. method`**

Returns *request*'s HTTP method, which is "`GET`" by default.

`request` **`. url`**

Returns the URL of *request* as a string.

`request` **`. headers`**

Returns a `Headers` object consisting of the headers associated with *request*. Note that headers added in the network layer by the user agent will not be accounted for in this object, e.g., the "`Host`" header.

`request` **`. destination`**

Returns the kind of resource requested by *request*, e.g., "`document`" or "`script`".

`request` **`. referrer`**

Returns the referrer of *request*. Its value can be a same-origin URL if explicitly set in *init*, the empty string to indicate no referrer, and "`about:client`" when defaulting to the global's default. This is used during fetching to determine the value of the `` `Referer` `` header of the request being made.

`request` **`. referrerPolicy`**

Returns the referrer policy associated with *request*. This is used during fetching to compute the value of the *request*'s referrer.

`request` **`. mode`**

Returns the mode associated with *request*, which is a string indicating whether the request will use CORS, or will be restricted to same-origin URLs.

`request` **`. credentials`**

Returns the credentials mode associated with *request*, which is a string indicating whether credentials will be sent with the request always, never, or only when sent to a same-origin URL.

`request` **`. cache`**

Returns the cache mode associated with *request*, which is a string indicating how the request will interact with the browser's cache when fetching.

`request` **`. redirect`**

Returns the redirect mode associated with *request*, which is a string indicating how redirects for the request will be handled during fetching. A request will follow redirects by default.

`request` **`. integrity`**

Returns *request*'s subresource integrity metadata, which is a cryptographic hash of the resource being fetched. Its value consists of multiple hashes separated by whitespace. [SRI]

`request` **`. keepalive`**

Returns a boolean indicating whether or not *request* can outlive the global in which it was created.

`request` **`. isReloadNavigation`**

Returns a boolean indicating whether or not *request* is for a reload navigation.

`request` **`. isHistoryNavigation`**

Returns a boolean indicating whether or not *request* is for a history navigation (a.k.a. back-foward navigation).

`request` **`. signal`**

File an issue about the selected text

Returns the signal associated with *request*, which is an `AbortSignal` object indicating whether or not *request* has been aborted, and its abort event handler.

The **`Request(input, init)`** constructor must run these steps:

1. Let *request* be null.
2. Let *fallbackMode* be null.
3. Let *fallbackCredentials* be null.
4. Let *baseURL* be current settings object's API base URL.
5. Let *signal* be null.
6. If *input* is a string, then:
    1. Let *parsedURL* be the result of parsing *input* with *baseURL*.
    2. If *parsedURL* is failure, then throw a `TypeError`.
    3. If *parsedURL* includes credentials, then throw a `TypeError`.
    4. Set *request* to a new request whose url is *parsedURL*.
    5. Set *fallbackMode* to "`cors`".
    6. Set *fallbackCredentials* to "`same-origin`".
7. Otherwise (*input* is a `Request` object):
    1. Set *request* to *input*'s request.
    2. Set *signal* to *input*'s signal.
8. Let *origin* be current settings object's origin.
9. Let *window* be "`client`".
10. If *request*'s window is an environment settings object and its origin is same origin with *origin*, set *window* to *request*'s window.
11. If *init*["`window`"] exists and is non-null, then throw a `TypeError`.
12. If *init*["`window`"] exists, then set *window* to "`no-window`".
13. Set *request* to a new request with the following properties:

    URL
    request's current URL.

    method
    request's method.

    header list
    A copy of *request*'s header list.

    unsafe-request flag
    Set.

    client
    Current settings object.

    window
    window.

    priority
    request's priority.

    origin
    "`client`".

    referrer
    request's referrer.

File an issue about the selected text

referrer policy
: *request*'s referrer policy.

mode
: *request*'s mode.

credentials mode
: *request*'s credentials mode.

cache mode
: *request*'s cache mode.

redirect mode
: *request*'s redirect mode.

integrity metadata
: *request*'s integrity metadata.

keepalive flag
: *request*'s keepalive flag.

reload-navigation flag
: *request*'s reload-navigation flag.

history-navigation flag
: *request*'s history-navigation flag.

14. If *init* is not empty, then:
    1. If *request*'s mode is "`navigate`", then set it to "`same-origin`".
    2. Unset *request*'s reload-navigation flag.
    3. Unset *request*'s history-navigation flag.
    4. Set *request*'s referrer to "`client`"
    5. Set *request*'s referrer policy to the empty string.

    Note

    This is done to ensure that when a service worker "redirects" a request, e.g., from an image in a cross-origin style sheet, and makes modifications, it no longer appears to come from the original source (i.e., the cross-origin style sheet), but instead from the service worker that "redirected" the request. This is important as the original source might not even be able to generate the same kind of requests as the service worker. Services that trust the original source could therefore be exploited were this not done, although that is somewhat farfetched.

15. If *init*["`referrer`"] exists, then:
    1. Let *referrer* be *init*["`referrer`"].
    2. If *referrer* is the empty string, then set *request*'s referrer to "`no-referrer`".
    3. Otherwise:
        1. Let *parsedReferrer* be the result of parsing *referrer* with *baseURL*.
        2. If *parsedReferrer* is failure, then throw a `TypeError`.
        3. If one of the following is true
            - *parsedReferrer*'s cannot-be-a-base-URL flag is set, scheme is "`about`", and path contains a single string "`client`"
            - *parsedReferrer*'s origin is not same origin with *origin*

            then set *request*'s referrer to "`client`".
        4. Otherwise, set *request*'s referrer to *parsedReferrer*.
16. If *init*["`referrerPolicy`"] exists, then set *request*'s referrer policy to it.
17. Let *mode* be *init*["`mode`"] if it exists, and *fallbackMode* otherwise.
18. If *mode* is "`navigate`", then throw a `TypeError`.
19. If *mode* is non-null, set *request*'s mode to *mode*.
20. Let *credentials* be *init*["`credentials`"] if it exists, and *fallbackCredentials* otherwise.
21. [illegible] set *request*'s credentials mode to *credentials*.

File an issue about the selected text

22. If *init*["cache"] exists, then set *request*'s cache mode to it.
23. If *request*'s cache mode is "only-if-cached" and *request*'s mode is *not* "same-origin", then throw a TypeError.
24. If *init*["redirect"] exists, then set *request*'s redirect mode to it.
25. If *init*["integrity"] exists, then set *request*'s integrity metadata to it.
26. If *init*["keepalive"] exists, then set *request*'s keepalive flag if *init*["keepalive"] is true, and unset it otherwise.
27. If *init*["method"] exists, then:
    1. Let *method* be *init*["method"].
    2. If *method* is not a method or *method* is a forbidden method, then throw a TypeError.
    3. Normalize *method*.
    4. Set *request*'s method to *method*.
28. If *init*["signal"] exists, then set *signal* to it.
29. Let *r* be a new Request object associated with *request*.
30. If *signal* is not null, then make *r*'s signal follow *signal*.
31. Set *r*'s headers to a new Headers object, whose header list is *request*'s header list, and guard is "request".
32. If *init* is not empty, then:

    Note

    The headers are sanitised as they might contain headers that are not allowed by this mode. Otherwise, they were previously sanitised or are unmodified since creation by a privileged API.

    1. Let *headers* be a copy of *r*'s headers and its associated header list.
    2. If *init*["headers"] exists, then set *headers* to *init*["headers"].
    3. Empty *r*'s headers's header list.
    4. If *r*'s request's mode is "no-cors", then:
        1. If *r*'s request's method is not a CORS-safelisted method, then throw a TypeError.
        2. Set *r*'s headers's guard to "request-no-cors".
    5. If *headers* is a Headers object, then for each *header* in its header list, append *header*'s name/*header*'s value to *r*'s Headers object.
    6. Otherwise, fill *r*'s Headers object with *headers*.
33. Let *inputBody* be *input*'s request's body if *input* is a Request object, and null otherwise.
34. If either *init*["body"] exists and is non-null or *inputBody* is non-null, and *request*'s method is `GET` or `HEAD`, then throw a TypeError.
35. Let *body* be *inputBody*.
36. If *init*["body"] exists and is non-null, then:
    1. Let *Content-Type* be null.
    2. If *init*["keepalive"] exists and is true, then set *body* and *Content-Type* to the result of extracting *init*["body"], with the *keepalive flag* set.
    3. Otherwise, set *body* and *Content-Type* to the result of extracting *init*["body"].
    4. If *Content-Type* is non-null and *r*'s headers's header list does not contain `Content-Type`, then append `Content-Type`/*Content-Type* to *r*'s headers.
37. If *body* is non-null and *body*'s source is null, then:
    1. If *r*'s request's mode is neither "same-origin" nor "cors", then throw a TypeError.
    2. Set *r*'s request's use-CORS-preflight flag.
38. If *inputBody* is *body* and *input* is disturbed or locked, then throw a TypeError.
39. If *inputBody* is *body* and *inputBody* is non-null, then:
    1. Let *rs* be a ReadableStream object from which one can read the exactly same data as one could read from *inputBody*'s stream.

File an issue about the selected text

This will be specified more precisely once transform stream and piping are precisely defined. See issue #463.

Note

This makes inputBody*'s stream locked and disturbed immediately.*

2. Set *body* to a new body whose stream is *rs*, whose source is *inputBody*'s source, and whose total bytes is *inputBody*'s total bytes.

40. Set *r*'s request's body to *body*.

41. Set *r*'s MIME type to the result of extracting a MIME type from *r*'s request's header list.

42. Return *r*.

The **`method`** attribute's getter, when invoked, must return the context object's request's method.

The **`url`** attribute's getter, when invoked, must return the context object's request's URL, serialized.

The **`headers`** attribute's getter, when invoked, must return the context object's headers.

The **`destination`** attribute's getter, when invoked, must return the context object's request's destination.

The **`referrer`** attribute's getter, when invoked, must return the empty string if the context object's request's referrer is "`no-referrer`", "`about:client`" if the context object's request's referrer is "`client`", and the context object's request's referrer, serialized, otherwise.

The **`referrerPolicy`** attribute's getter, when invoked, must return the context object's request's referrer policy.

The **`mode`** attribute's getter, when invoked, must return the context object's request's mode.

The **`credentials`** attribute's getter, when invoked, must return the context object's request's credentials mode.

The **`cache`** attribute's getter, when invoked, must return the context object's request's cache mode.

The **`redirect`** attribute's getter, when invoked, must return the context object's request's redirect mode.

The **`integrity`** attribute's getter, when invoked, must return the context object's request's integrity metadata.

The **`keepalive`** attribute's getter, when invoked, must return true if the context object's request's keepalive flag is set, and false otherwise.

The **`isReloadNavigation`** attribute's getter, when invoked, must return true if the context object's request's reload-navigation flag is set, and false otherwise.

The **`isHistoryNavigation`** attribute's getter, when invoked, must return true if the context object's request's history-navigation flag is set, and false otherwise.

The **`signal`** attribute's getter, when invoked, must return the context object's signal.

The **`clone()`** method, when invoked, must run these steps:

1. If the context object is disturbed or locked, then throw a `TypeError`.
2. Let *clonedRequestObject* be a new `Request` object.
3. Let *clonedRequest* be the result of cloning the context object's request.
4. Set *clonedRequestObject*'s request to *clonedRequest*.
5. Set *clonedRequestObject*'s headers to a new `Headers` object with the following properties:

   header list

   clonedRequest's header list.

   guard

   The context object's headers's guard.
6. Make *clonedRequestObject*'s signal follow the context object's signal.
7. Return *clonedRequestObject*.

File an issue about the selected text

5.4. Response class §

IDL
```
[Constructor(optional BodyInit? body = null, optional ResponseInit init), Exposed=(Window,Worker)]
interface Response {
  [NewObject] static Response error();
  [NewObject] static Response redirect(USVString url, optional unsigned short status = 302);

  readonly attribute ResponseType type;

  readonly attribute USVString url;
  readonly attribute boolean redirected;
  readonly attribute unsigned short status;
  readonly attribute boolean ok;
  readonly attribute ByteString statusText;
  [SameObject] readonly attribute Headers headers;
  readonly attribute Promise<Headers> trailer;

  [NewObject] Response clone();
};
Response includes Body;

dictionary ResponseInit {
  unsigned short status = 200;
  ByteString statusText = "";
  HeadersInit headers;
};

enum ResponseType { "basic", "cors", "default", "error", "opaque", "opaqueredirect" };
```

A `Response` object has an associated **response** (a response).

A `Response` object also has an associated **headers** (null or a `Headers` object), initially null.

A `Response` object also has an associated **trailer promise** (a promise). Note *Used for the `trailer` attribute.*

A `Response` object's body is its response's body.

The **`Response(body, init)`** constructor, when invoked, must run these steps:

1. If *init*["`status`"] is not in the range `200` to `599`, inclusive, then throw a `RangeError`.
2. If *init*["`statusText`"] does not match the reason-phrase token production, then throw a `TypeError`.
3. Let *r* be a new `Response` object associated with a new response.
4. Set *r*'s headers to a new `Headers` object, whose header list is *r*'s response's header list, and guard is "`response`".
5. Set *r*'s response's status to *init*["`status`"].
6. Set *r*'s response's status message to *init*["`statusText`"].
7. If *init*["`headers`"] exists, then fill *r*'s headers with *init*["`headers`"].
8. If *body* is non-null, then:
    1. If *init*["`status`"] is a null body status, then throw a `TypeError`.

       Note

       `101` is included in null body status due to its use elsewhere. It does not affect this step.
    2. Let *Content-Type* be null.
    3. Set *r*'s response's body and *Content-Type* to the result of extracting *body*.
    4. If *Content-Type* is non-null and *r*'s response's header list does not contain `` `Content-Type` ``, then append `` `Content-Type` ``/*Content-Type* to *r*'s response's header list.
9. Set *r*'s MIME type to the result of extracting a MIME type from *r*'s response's header list.
10. Set *r*'s response's HTTPS state to current settings object's HTTPS state.
11. Resolve *r*'s trailer promise with a new `Headers` object whose guard is "`immutable`".

File an issue about the selected text

12. Return *r*.

The static **`error()`** method, when invoked, must run these steps:

1. Let *r* be a new `Response` object, whose response is a new network error.
2. Set *r*'s headers to a new `Headers` object whose guard is "`immutable`".
3. Return *r*.

The static **`redirect(url, status)`** method, when invoked, must run these steps:

1. Let *parsedURL* be the result of parsing *url* with current settings object's API base URL.
2. If *parsedURL* is failure, then throw a `TypeError`.
3. If *status* is not a redirect status, then throw a `RangeError`.
4. Let *r* be a new `Response` object, whose response is a new response.
5. Set *r*'s headers to a new `Headers` object whose guard is "`immutable`".
6. Set *r*'s response's status to *status*.
7. Append `` `Location` `` to *parsedURL*, serialized and isomorphic encoded, in *r*'s response's header list.
8. Return *r*.

The **`type`** attribute's getter, when invoked, must return the context object's response's type.

The **`url`** attribute's getter, when invoked, must return the empty string if the context object's response's URL is null and the context object's response's URL, serialized with the *exclude-fragment flag* set, otherwise. [URL]

The **`redirected`** attribute's getter, when invoked, must return true if the context object's response's URL list has more than one item, and false otherwise.

Note

To filter out responses that are the result of a redirect, do this directly through the API, e.g., `fetch(url, { redirect:"error" })`. This way a potentially unsafe response cannot accidentally leak.

The **`status`** attribute's getter, when invoked, must return the context object's response's status.

The **`ok`** attribute's getter, when invoked, must return true if the context object's response's status is an ok status, and false otherwise.

The **`statusText`** attribute's getter, when invoked, must return the context object's response's status message.

The **`headers`** attribute's getter, when invoked, must return the context object's headers.

The **`trailer`** attribute's getter, when invoked, must return the context object's trailer promise.

The **`clone()`** method, when invoked, must run these steps:

1. If the context object is disturbed or locked, then throw a `TypeError`.
2. Let *clonedResponseObject* be a new `Response` object.
3. Let *clonedResponse* be the result of cloning the context object's response.
4. Set *clonedResponseObject*'s response to *clonedResponse*.
5. Set *clonedResponseObject*'s headers to a new `Headers` object whose header list is set to *clonedResponse*'s header list, and guard is the context object's headers's guard.
6. Upon fulfillment of the context object's trailer promise, resolve *clonedResponseObject*'s trailer promise with a new `Headers` object whose guard is "`immutable`", and whose header list is *clonedResponse*'s trailer.
7. Return *clonedResponseObject*.
8. Return *clonedResponse*.

File an issue about the selected text

5.5. Fetch method §

```
IDL
partial interface mixin WindowOrWorkerGlobalScope {
  [NewObject] Promise<Response> fetch(RequestInfo input, optional RequestInit init);
};
```

The **`fetch(input, init)`** method, must run these steps:

1. Let *p* be a new promise.
2. Let *requestObject* be the result of invoking the initial value of `Request` as constructor with *input* and *init* as arguments. If this throws an exception, reject *p* with it and return *p*.
3. Let *request* be *requestObject*'s request.
4. If *requestObject*'s signal's aborted flag is set, then:
    1. Abort fetch with *p*, *request*, and null.
    2. Return *p*.
5. If *request*'s client's global object is a `ServiceWorkerGlobalScope` object, then set *request*'s service-workers mode to "`none`".
6. Let *responseObject* be a new `Response` object and a new associated `Headers` object whose guard is "`immutable`".
7. Let *locallyAborted* be false.

    Note

    This lets us reject promises with predictable timing, when the request to abort comes from the same thread as the call to fetch.
8. Add the following abort steps to *requestObject*'s signal:
    1. Set *locallyAborted* to true.
    2. Abort fetch with *p*, *request*, and *responseObject*.
    3. Terminate the ongoing fetch with the aborted flag set.
9. Run the following in parallel:

    Fetch *request*.

    To process response for *response*, run these substeps:
    1. If *locallyAborted* is true, terminate these substeps.
    2. If *response*'s aborted flag is set, then abort fetch with *p*, *request*, and *responseObject*, and terminate these substeps.
    3. If *response* is a network error, then reject *p* with a `TypeError` and terminate these substeps.
    4. Associate *responseObject* with *response*.
    5. Resolve *p* with *responseObject*.

    To process response done for *response*, run these substeps:
    1. If *locallyAborted* is true, terminate these substeps.
    2. Let *trailerObject* be a new `Headers` object whose guard is "`immutable`".
    3. If *response*'s trailer failed flag is set, then:
        1. If *response*'s aborted flag is set, reject *responseObject*'s trailer promise with an "`AbortError`" `DOMException`.
        2. Otherwise, reject *responseObject*'s trailer promise with a `TypeError`.
        3. Terminate these substeps.
    4. Associate *trailerObject* with *response*'s trailer.
    5. Resolve *responseObject*'s trailer promise with *trailerObject*.
10. Return *p*.

To **abort fetch** with a *promise*, *request*, and *responseObject*, run these steps:

1. Let *error* be an "`AbortError`" `DOMException`.

File an issue about the selected text

2. Reject *promise* with *error*.

   Note

   This is a no-op if promise *has already fulfilled.*

3. If *request*'s body is not null and is readable, then cancel *request*'s body with *error*.

4. If *responseObject* is null, then return.

5. Reject *responseObject*'s trailer promise with *error*.

   Note

   This is a no-op if responseObject*'s trailer promise has already fulfilled.*

6. Let *response* be *responseObject*'s response.

7. If *response*'s body is not null and is readable, then error *response*'s body with *error*.

5.6. Garbage collection §

The user agent may terminate an ongoing fetch if that termination is not observable through script.

Note

"Observable through script" means observable through `fetch()`'s arguments and return value. Other ways, such as communicating with the server through a side-channel are not included.

Note

The server being able to observe garbage collection has precedent, e.g., with `WebSocket` and `XMLHttpRequest` objects.

Example

The user agent can terminate the fetch because the termination cannot be observed.

```
fetch("https://www.example.com/")
```

The user agent cannot terminate the fetch because the termination can be observed through the promise.

```
window.promise = fetch("https://www.example.com/")
```

The user agent can terminate the fetch because the associated body is not observable.

```
window.promise = fetch("https://www.example.com/").then(res => res.headers)
```

The user agent can terminate the fetch because the termination cannot be observed.

```
fetch("https://www.example.com/").then(res => res.body.getReader().closed)
```

The user agent cannot terminate the fetch because one can observe the termination by registering a handler for the promise object.

```
window.promise = fetch("https://www.example.com/")
  .then(res => res.body.getReader().closed)
```

The user agent cannot terminate the fetch as termination would be observable via the registered handler.

```
fetch("https://www.example.com/")
  .then(res => {
    res.body.getReader().closed.then(() => console.log("stream closed!"))
  })
```

(The above examples of non-observability assume that built-in properties and functions, such as `body.getReader()`, have not been overwritten.)

File an issue about the selected text

6. WebSocket protocol alterations §

Note

This section replaces part of the WebSocket protocol opening handshake client requirement to integrate it with algorithms defined in Fetch. This way CSP, cookies, HSTS, and other Fetch-related protocols are handled in a single location. Ideally the RFC would be updated with this language, but it is never that easy. The WebSocket API, defined in the HTML Standard, has been updated to use this language. [WSP] [HTML]

The way this works is by replacing The WebSocket Protocol's "establish a WebSocket connection" algorithm with a new one that integrates with Fetch. "Establish a WebSocket connection" consists of three algorithms: setting up a connection, creating and transmiting a handshake request, and validating the handshake response. That layering is different from Fetch, which first creates a handshake, then sets up a connection and transmits the handshake, and finally validates the response. Keep that in mind while reading these alterations.

6.1. Connections §

To **obtain a WebSocket connection**, given a *url*, run these steps:

1. Let *host* be *url*'s host.
2. Let *port* be *url*'s port.
3. Let *secure* be false, if *url*'s scheme is "`http`", and true otherwise.
4. Follow the requirements stated in step 2 to 5, inclusive, of the first set of steps in section 4.1 of The WebSocket Protocol to establish a WebSocket connection. [WSP]
5. If that established a connection, return it, and return failure otherwise.

Note

Although structured a little differently, carrying different properties, and therefore not shareable, a WebSocket connection is very close to identical to an "ordinary" connection.

6.2. Opening handshake §

To **establish a WebSocket connection**, given a *url*, *protocols*, and *client*, run these steps:

1. Let *requestURL* be a copy of *url*, with its scheme set to "`http`", if *url*'s scheme is "`ws`", and to "`https`" otherwise.

   Note

   This change of scheme is essential to integrate well with fetching. E.g., HSTS would not work without it. There is no real reason for WebSocket to have distinct schemes, it's a legacy artefact. [HSTS]

2. Let *request* be a new request, whose URL is *requestURL*, client is *client*, service-workers mode is "`none`", referrer is "`no-referrer`", synchronous flag is set, mode is "`websocket`", credentials mode is "`include`", cache mode is "`no-store`", and redirect mode is "`error`".
3. Append `` `Upgrade` ``/`` `websocket` `` to *request*'s header list.
4. Append `` `Connection` ``/`` `Upgrade` `` to *request*'s header list.
5. Let *keyValue* be a nonce consisting of a randomly selected 16-byte value that has been forgiving-base64-encoded and isomorphic encoded.

   ¶ Example

   If the randomly selected value was the byte sequence 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x0f 0x10, *keyValue* would be forgiving-base64-encoded to "`AQIDBAUGBwgJCgsMDQ4PEC==`" and isomorphic encoded to `` `AQIDBAUGBwgJCgsMDQ4PEC==` ``.

6. Append `` `Sec-WebSocket-Key` ``/*keyValue* to *request*'s header list.

File an issue about the selected text et-Version`/`13` to *request*'s header list.

8. For each *protocol* in *protocols*, combine `` `Sec-WebSocket-Protocol` ``/*protocol* in *request*'s header list.
9. Let *permessageDeflate* be a user-agent defined "`permessage-deflate`" extension header value. [WSP]

¶ Example

```
`permessage-deflate; client_max_window_bits`
```

10. Append `` `Sec-WebSocket-Extensions` ``/*permessageDeflate* to *request*'s header list.
11. Let *response* be the result of fetching *request*.
12. If *response* is a network error or its status is not `101`, fail the WebSocket connection.
13. If *protocols* is not the empty list and extracting header list values given `` `Sec-WebSocket-Protocol` `` and *response*'s header list results in null, failure, or the empty byte sequence, then fail the WebSocket connection.

    Note

    This is different from the check on this header defined by The WebSocket Protocol. That only covers a subprotocol not requested by the client. This covers a subprotocol requested by the client, but not acknowledged by the server.

14. Follow the requirements stated step 2 to step 6, inclusive, of the last set of steps in section 4.1 of The WebSocket Protocol to validate *response*. This either results in fail the WebSocket connection or the WebSocket connection is established.

Fail the WebSocket connection and **the WebSocket connection is established** are defined by The WebSocket Protocol. [WSP]

⚠Warning!

The reason redirects are not followed and this handshake is generally restricted is because it could introduce serious security problems in a web browser context. For example, consider a host with a WebSocket server at one path and an open HTTP redirector at another. Suddenly, any script that can be given a particular WebSocket URL can be tricked into communicating to (and potentially sharing secrets with) any host on the internet, even if the script checks that the URL has the right hostname.

File an issue about the selected text

7. `data:` URLs §

For an informative description of `data:` URLs, see RFC 2397. This section replaces that RFC's normative processing requirements to be compatible with deployed content. [RFC2397]

A **`data:` URL struct** is a struct that consists of a **MIME type** (a MIME type) and a **body** (a byte sequence).

The **`data:` URL processor** takes a URL *dataURL* and then runs these steps:

1. Assert: *dataURL*'s scheme is "`data`".
2. Let *input* be the result of running the URL serializer on *dataURL* with the *exclude fragment flag* set.
3. Remove the leading "`data:`" string from *input*.
4. Let *position* point at the start of *input*.
5. Let *mimeType* be the result of collecting a sequence of code points that are not equal to U+002C (,), given *position*.
6. Strip leading and trailing ASCII whitespace from *mimeType*.

   Note

   This will only remove U+0020 SPACE code points, if any.
7. If *position* is past the end of *input*, then return failure.
8. Advance *position* by 1.
9. Let *encodedBody* be the remainder of *input*.
10. Let *body* be the string percent decoding of *encodedBody*.
11. If *mimeType* ends with U+003B (;), followed by zero or more U+0020 SPACE, followed by an ASCII case-insensitive match for "`base64`", then:
    1. Let *stringBody* be the isomorphic decode of *body*.
    2. Set *body* to the forgiving-base64 decode of *stringBody*.
    3. If *body* is failure, then return failure.
    4. Remove the last 6 code points from *mimeType*.
    5. Remove trailing U+0020 SPACE code points from *mimeType*, if any.
    6. Remove the last U+003B (;) code point from *mimeType*.
12. If *mimeType* starts with U+003B (;), then prepend "`text/plain`" to *mimeType*.
13. Let *mimeTypeRecord* be the result of parsing *mimeType*.
14. If *mimeTypeRecord* is failure, then set *mimeTypeRecord* to `text/plain;charset=US-ASCII`.
15. Return a new `data:` URL struct whose MIME type is *mimeTypeRecord* and body is *body*.

File an issue about the selected text

Background reading §

This section and its subsections are informative only.

HTTP header layer division

For the purposes of fetching, there is an API layer (HTML's `img`, CSS' `background-image`), early fetch layer, service worker layer, and network & cache layer. `` `Accept` `` and `` `Accept-Language` `` are set in the early fetch layer (typically by the user agent). Most other headers controlled by the user agent, such as `` `Accept-Encoding` ``, `` `Host` ``, and `` `Referer` ``, are set in the network & cache layer. Developers can set headers either at the API layer or in the service worker layer (typically through a `Request` object). Developers have almost no control over forbidden headers, but can control `` `Accept` `` and have the means to constrain and omit `` `Referer` `` for instance.

Atomic HTTP redirect handling

Redirects (a response whose status or internal response's (if any) status is a redirect status) are not exposed to APIs. Exposing redirects might leak information not otherwise available through a cross-site scripting attack.

Example

A fetch to `https://example.org/auth` that includes a `Cookie` marked `HttpOnly` could result in a redirect to `https://other-origin.invalid/4af955781ea1c84a3b11`. This new URL contains a secret. If we expose redirects that secret would be available through a cross-site scripting attack.

Basic safe CORS protocol setup §

For resources where data is protected through IP authentication or a firewall (unfortunately relatively common still), using the CORS protocol is **unsafe**. (This is the reason why the CORS protocol had to be invented.)

However, otherwise using the following header is **safe**:

```
Access-Control-Allow-Origin: *
```

Even if a resource exposes additional information based on cookie or HTTP authentication, using the above header will not reveal it. It will share the resource with APIs such as `XMLHttpRequest`, much like it is already shared with `curl` and `wget`.

Thus in other words, if a resource cannot be accessed from a random device connected to the web using `curl` and `wget` the aforementioned header is not to be included. If it can be accessed however, it is perfectly fine to do so.

CORS protocol and HTTP caches §

If CORS protocol requirements are more complicated than setting `` `Access-Control-Allow-Origin` `` to `*` or a static origin, `` `Vary` `` is to be used. [HTML] [HTTP] [HTTP-SEMANTICS] [HTTP-COND] [HTTP-CACHING] [HTTP-AUTH]

Example

```
Vary: Origin
```

In particular, consider what happens if `` `Vary` `` is *not* used and a server is configured to send `` `Access-Control-Allow-Origin` `` for a certain resource only in response to a CORS request. When a user agent receives a response to a non-CORS request for that resource (for example, as the result of a navigation request), the response will lack `` `Access-Control-Allow-Origin` `` and the user agent will cache that response. Then, if the user agent subsequently encounters a CORS request for the resource, it will use that cached response from the previous non-CORS request, without `` `Access-Control-Allow-Origin` ``.

File an issue about the selected text

But if `Vary: Origin` is used in the same scenario described above, it will cause the user agent to fetch a response that includes `Access-Control-Allow-Origin`, rather than using the cached response from the previous non-CORS request that lacks `Access-Control-Allow-Origin`.

However, if `Access-Control-Allow-Origin` is set to * or a static origin for a particular resource, then configure the server to always send `Access-Control-Allow-Origin` in responses for the resource — for non-CORS requests as well as CORS requests — and do not use `Vary`.

File an issue about the selected text

Acknowledgments §

Thanks to Adam Barth, Adam Lavin, Alan Jeffrey, Alexey Proskuryakov, Andrés Gutiérrez, Andrew Sutherland, Ángel González, Anssi Kostiainen, Arkadiusz Michalski, Arne Johannessen, Arthur Barstow, Asanka Herath, Axel Rauschmayer, Ben Kelly, Benjamin Gruenbaum, Benjamin Hawkes-Lewis, Bert Bos, Björn Höhrmann, Boris Zbarsky, Brad Hill, Brad Porter, Bryan Smith, Caitlin Potter, Cameron McCormack, Chris Rebert, Clement Pellerin, Collin Jackson, Daniel Robertson, Daniel Veditz, David Benjamin, David Håsäther, David Orchard, Dean Jackson, Devdatta Akhawe, Domenic Denicola, Dominic Farolino, Dominique Hazaël-Massieux, Doug Turner, Douglas Creager, Eero Häkkinen, Ehsan Akhgari, Emily Stark, Eric Lawrence, François Marier, Frank Ellerman, Frederick Hirsch, Gary Blackwood, Gavin Carothers, Glenn Maynard, Graham Klyne, Hal Lockhart, Hallvord R. M. Steen, Harris Hancock, Henri Sivonen, Henry Story, Hiroshige Hayashizaki, Honza Bambas, Ian Hickson, Ilya Grigorik, isonmad, Jake Archibald, James Graham, Janusz Majnert, Jeena Lee, Jeff Carpenter, Jeff Hodges, Jeffrey Yasskin, Jesse M. Heines, Jianjun Chen, Jinho Bang, Jochen Eisinger, John Wilander, Jonas Sicking, Jonathan Kingston, Jonathan Watt, 최종찬 (Jongchan Choi), Jörn Zaefferer, Joseph Pecoraro, Josh Matthews, Julian Krispel-Samsel, Julian Reschke, 송정기 (Jungkee Song), Jussi Kalliokoski, Jxck, Kagami Sascha Rosylight, Keith Yeung, Kenji Baheux, Lachlan Hunt, Larry Masinter, Liam Brummitt, Louis Ryan, Lucas Gonze, Łukasz Anforowicz, 呂康豪 (Kang-Hao Lu), Maciej Stachowiak, Malisa, Manfred Stock, Manish Goregaokar, Marc Silbey, Marcos Caceres, Marijn Kruisselbrink, Mark Nottingham, Mark S. Miller, Martin Dürst, Martin Thomson, Matt Andrews, Matt Falkenhagen, Matt Oshry, Matt Seddon, Matt Womer, Mhano Harkness, Michael Kohler, Michael™ Smith, Mike Pennisi, Mike West, Mohamed Zergaoui, Mohammed Zubair Ahmed, Moritz Kneilmann, Ms2ger, Nico Schlömer, Nikhil Marathe, Nikki Bee, Nikunj Mehta, Odin Hørthe Omdal, Ondřej Žára, O. Opsec, Philip Jägenstedt, R. Auburn, Raphael Kubo da Costa, Rondinelly, Rory Hewitt, Ryan Sleevi, Sébastien Cevey, Sendil Kumar N, Shao-xuan Kang, Sharath Udupa, Shivakumar Jagalur Matt, Sigbjørn Finne, Simon Pieters, Simon Sapin, Srirama Chandra Sekhar Mogali, Stephan Paul, Steven Salat, Sunava Dutta, Surya Ismail, Takashi Toyoshima, 吉野剛史 (Takeshi Yoshino), Thomas Roessler, Thomas Steiner, Thomas Wisniewski, Tiancheng "Timothy" Gu, Tobie Langel, Tom Schuster, Tomás Aparicio, 保呂毅 (Tsuyoshi Horo), Tyler Close, Ujjwal Sharma, Vignesh Shanmugam, Vladimir Dzhuvinov, Wayne Carr, Xabier Rodríguez, Yehuda Katz, Yoav Weiss, Youenn Fablet, 平野裕 (Yutaka Hirano), and Zhenbin Xu for being awesome.

This standard is written by Anne van Kesteren (Mozilla, annevk@annevk.nl).

File an issue about the selected text

Index §

Terms defined by this specification §

File an issue about the selected text

File an issue about the selected text

File an issue about the selected text

File an issue about the selected text

File an issue about the selected text

File an issue about the selected text

Terms defined by reference §

- [DOM] defines the following terms:
  - AbortSignal
  - aborted flag
  - add
  - context object
  - follow
- [ENCODING] defines the following terms:
  - utf-8
  - utf-8 decode
  - utf-8 encode
- [FILEAPI] defines the following terms:
  - Blob
  - File
  - name
  - object
  - read operation
  - size
  - type
- [HTML] defines the following terms:
  - WebSocket
  - Window
  - WindowOrWorkerGlobalScope
  - active document
  - ancestor browsing context
  - api base url
  - ascii serialization of an origin
  - creation url
  - current settings object
  - downloads a hyperlink
  - environment
  - environment settings object
  - form
  - global object
  - https state
  - id
  - in parallel
  - multipart/form-data boundary string
  - multipart/form-data encoding algorithm
  - navigate
  - nested browsing context
  - networking task source
  - opaque origin
  - origin (for environment settings object)
  - queue a task
  - referrer policy
  - resource fetch algorithm
  - responsible browsing context
  - responsible event loop
  - same origin
  - target browsing context
  - task
- [HTTP] defines the following terms:
  - field-name
  - method
  - reason-phrase
- [HTTP-CACHING] defines the following terms:
  - delta-seconds
- [INFRA] defines the following terms:
  - abort when
  - append
  - ascii case-insensitive
  - ascii whitespace
  - break
  - byte sequence
  - byte-case-insensitive
  - byte-lowercase
  - byte-uppercase

File an issue about the selected text

- clone
- code point
- collecting a sequence of code points
- contain
- continue
- exist
- for each (for map)
- forgiving-base64 decode
- forgiving-base64 encode
- if aborted
- is empty
- is not empty (for map)
- isomorphic decode
- isomorphic encode
- item
- length
- list
- parse json from bytes
- position variable
- remove
- string
- strip leading and trailing ascii whitespace
- struct

- [MIMESNIFF] defines the following terms:
  - essence
  - html mime type
  - javascript mime type
  - json mime type
  - mime type
  - parameters
  - parse a mime type
  - serialize a mime type
  - serialize a mime type to bytes
  - xml mime type
- [promises-guide] defines the following terms:
  - promise-calling
- [REFERRER] defines the following terms:
  - ReferrerPolicy
  - determine request's referrer
  - referrer policy
  - set request's referrer policy on redirect
- [STREAMS] defines the following terms:
  - AcquireReadableStreamDefaultReader
  - CreateReadableStream
  - IsReadableStreamDisturbed
  - IsReadableStreamLocked
  - ReadableStreamCancel
  - ReadableStreamDefaultControllerClose
  - ReadableStreamDefaultControllerEnqueue
  - ReadableStreamDefaultControllerError
  - ReadableStreamDefaultControllerGetDesiredSize
  - ReadableStreamDefaultReaderRead
  - ReadableStreamTee
  - chunk
  - getReader({ mode } = {})
- [SW] defines the following terms:
  - ServiceWorkerGlobalScope
  - fetch
  - handle fetch
- [URL] defines the following terms:
  - URLSearchParams
  - blob url entry
  - cannot-be-a-base-url flag
  - domain
  - fragment
  - host
  - include credentials
  - list
  - origin

File an issue about the selected text

- path
- port
- same site
- scheme
- set the password
- set the username
- string percent decode
- url
- url parser
- url serializer
- urlencoded parser
- urlencoded serializer

- [WEBIDL] defines the following terms:
  - AbortError
  - ArrayBuffer
  - BufferSource
  - ByteString
  - DOMException
  - DOMString
  - Exposed
  - NewObject
  - RangeError
  - SameObject
  - TypeError
  - USVString
  - boolean
  - record
  - sequence
  - throw
  - unsigned short
  - value pairs to iterate over
- [XHR] defines the following terms:
  - FormData
  - XMLHttpRequest
  - XMLHttpRequestUpload
  - entries
  - entry

File an issue about the selected text

References §

Normative References §

[ABNF]

D. Crocker, Ed.; P. Overell. Augmented BNF for Syntax Specifications: ABNF. January 2008. Internet Standard. URL: https://tools.ietf.org/html/rfc5234

[COOKIES]

A. Barth. HTTP State Management Mechanism. April 2011. Proposed Standard. URL: https://tools.ietf.org/html/rfc6265

[CSP]

Mike West. Content Security Policy Level 3. URL: https://w3c.github.io/webappsec-csp/

[DOM]

Anne van Kesteren. DOM Standard. Living Standard. URL: https://dom.spec.whatwg.org/

[ENCODING]

Anne van Kesteren. Encoding Standard. Living Standard. URL: https://encoding.spec.whatwg.org/

[FILEAPI]

Marijn Kruisselbrink; Arun Ranganathan. File API. URL: https://w3c.github.io/FileAPI/

[HSTS]

J. Hodges; C. Jackson; A. Barth. HTTP Strict Transport Security (HSTS). November 2012. Proposed Standard. URL: https://tools.ietf.org/html/rfc6797

[HTML]

Anne van Kesteren; et al. HTML Standard. Living Standard. URL: https://html.spec.whatwg.org/multipage/

[HTTP]

R. Fielding, Ed.; J. Reschke, Ed.. Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. June 2014. Proposed Standard. URL: https://tools.ietf.org/html/rfc7230

[HTTP-AUTH]

R. Fielding, Ed.; J. Reschke, Ed.. Hypertext Transfer Protocol (HTTP/1.1): Authentication. June 2014. Proposed Standard. URL: https://tools.ietf.org/html/rfc7235

[HTTP-CACHING]

R. Fielding, Ed.; M. Nottingham, Ed.; J. Reschke, Ed.. Hypertext Transfer Protocol (HTTP/1.1): Caching. June 2014. Proposed Standard. URL: https://tools.ietf.org/html/rfc7234

[HTTP-COND]

R. Fielding, Ed.; J. Reschke, Ed.. Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests. June 2014. Proposed Standard. URL: https://tools.ietf.org/html/rfc7232

[HTTP-SEMANTICS]

R. Fielding, Ed.; J. Reschke, Ed.. Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. June 2014. Proposed Standard. URL: https://tools.ietf.org/html/rfc7231

[INFRA]

Anne van Kesteren; Domenic Denicola. Infra Standard. Living Standard. URL: https://infra.spec.whatwg.org/

[MIMESNIFF]

Gordon P. Hemsley. MIME Sniffing Standard. Living Standard. URL: https://mimesniff.spec.whatwg.org/

[MIX]

Mike West. Mixed Content. URL: https://w3c.github.io/webappsec-mixed-content/

[PROMISES-GUIDE]

Domenic Denicola. Writing Promise-Using Specifications. 16 February 2016. TAG Finding. URL: https://www.w3.org/2001/tag/doc/promises-guide

[REFERRER]

Jochen Eisinger; Emily Stark. Referrer Policy. URL: https://w3c.github.io/webappsec-referrer-policy/

[RFC7405]

P. Kyzivat. Case-Sensitive String Support in ABNF. December 2014. Proposed Standard. URL: https://tools.ietf.org/html/rfc7405

File an issue about the selected text

L. Masinter. Returning Values from Forms: multipart/form-data. July 2015. Proposed Standard. URL: https://tools.ietf.org/html/rfc7578

[RFC959]

J. Postel; J. Reynolds. File Transfer Protocol. October 1985. Internet Standard. URL: https://tools.ietf.org/html/rfc959

[SRI]

Devdatta Akhawe; et al. Subresource Integrity. URL: https://w3c.github.io/webappsec-subresource-integrity/

[STREAMS]

Adam Rice; Domenic Denicola; 吉野剛史 (Takeshi Yoshino). Streams Standard. Living Standard. URL: https://streams.spec.whatwg.org/

[SW]

Alex Russell; et al. Service Workers 1. URL: https://w3c.github.io/ServiceWorker/

[TLS]

E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. August 2018. Proposed Standard. URL: https://tools.ietf.org/html/rfc8446

[UPGRADE]

Mike West. Upgrade Insecure Requests. URL: https://w3c.github.io/webappsec-upgrade-insecure-requests/

[URL]

Anne van Kesteren. URL Standard. Living Standard. URL: https://url.spec.whatwg.org/

[WEBIDL]

Boris Zbarsky. Web IDL. URL: https://heycam.github.io/webidl/

[WSP]

I. Fette; A. Melnikov. The WebSocket Protocol. December 2011. Proposed Standard. URL: https://tools.ietf.org/html/rfc6455

[XHR]

Anne van Kesteren. XMLHttpRequest Standard. Living Standard. URL: https://xhr.spec.whatwg.org/

Informative References §

[CORS]

Anne van Kesteren. Cross-Origin Resource Sharing. 16 January 2014. REC. URL: https://www.w3.org/TR/cors/

[EXPECT-CT]

Emily Stark. Expect-CT Extension for HTTP. URL: https://tools.ietf.org/html/draft-ietf-httpbis-expect-ct-02

[HTTP-RANGE]

R. Fielding, Ed.; Y. Lafon, Ed.; J. Reschke, Ed.. Hypertext Transfer Protocol (HTTP/1.1): Range Requests. June 2014. Proposed Standard. URL: https://tools.ietf.org/html/rfc7233

[HTTPVERBSEC1]

Multiple vendors' web servers enable HTTP TRACE method by default.. URL: https://www.kb.cert.org/vuls/id/867593

[HTTPVERBSEC2]

Microsoft Internet Information Server (IIS) vulnerable to cross-site scripting via HTTP TRACK method.. URL: https://www.kb.cert.org/vuls/id/288308

[HTTPVERBSEC3]

HTTP proxy default configurations allow arbitrary TCP connections.. URL: https://www.kb.cert.org/vuls/id/150227

[OCSP]

S. Santesson; et al. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. June 2013. Proposed Standard. URL: https://tools.ietf.org/html/rfc6960

[ORIGIN]

A. Barth. The Web Origin Concept. December 2011. Proposed Standard. URL: https://tools.ietf.org/html/rfc6454

[RFC2397]

L. Masinter. The "data" URL scheme. August 1998. Proposed Standard. URL: https://tools.ietf.org/html/rfc2397

File an issue about the selected text

IDL Index §

```
typedef (sequence<sequence<ByteString>> or record<ByteString, ByteString>) HeadersInit;

[Constructor(optional HeadersInit init),
 Exposed=(Window,Worker)]
interface Headers {
  void append(ByteString name, ByteString value);
  void delete(ByteString name);
  ByteString? get(ByteString name);
  boolean has(ByteString name);
  void set(ByteString name, ByteString value);
  iterable<ByteString, ByteString>;
};
typedef (Blob or BufferSource or FormData or URLSearchParams or ReadableStream or USVString)
BodyInit;
interface mixin Body {
  readonly attribute ReadableStream? body;
  readonly attribute boolean bodyUsed;
  [NewObject] Promise<ArrayBuffer> arrayBuffer();
  [NewObject] Promise<Blob> blob();
  [NewObject] Promise<FormData> formData();
  [NewObject] Promise<any> json();
  [NewObject] Promise<USVString> text();
};
typedef (Request or USVString) RequestInfo;

[Constructor(RequestInfo input, optional RequestInit init),
 Exposed=(Window,Worker)]
interface Request {
  readonly attribute ByteString method;
  readonly attribute USVString url;
  [SameObject] readonly attribute Headers headers;

  readonly attribute RequestDestination destination;
  readonly attribute USVString referrer;
  readonly attribute ReferrerPolicy referrerPolicy;
  readonly attribute RequestMode mode;
  readonly attribute RequestCredentials credentials;
  readonly attribute RequestCache cache;
  readonly attribute RequestRedirect redirect;
  readonly attribute DOMString integrity;
  readonly attribute boolean keepalive;
  readonly attribute boolean isReloadNavigation;
  readonly attribute boolean isHistoryNavigation;
  readonly attribute AbortSignal signal;

  [NewObject] Request clone();
};
Request includes Body;

dictionary RequestInit {
  ByteString method;
  HeadersInit headers;
  BodyInit? body;
  USVString referrer;
  ReferrerPolicy referrerPolicy;
  RequestMode mode;
  RequestCredentials credentials;
  RequestCache cache;
  RequestRedirect redirect;
  DOMString integrity;
```

File an issue about the selected text

```
  AbortSignal? signal;
  any window; // can only be set to null
};

enum RequestDestination { "", "audio", "audioworklet", "document", "embed", "font", "image",
"manifest", "object", "paintworklet", "report", "script", "sharedworker", "style",  "track", "video",
"worker", "xslt" };
enum RequestMode { "navigate", "same-origin", "no-cors", "cors" };
enum RequestCredentials { "omit", "same-origin", "include" };
enum RequestCache { "default", "no-store", "reload", "no-cache", "force-cache", "only-if-cached" };
enum RequestRedirect { "follow", "error", "manual" };
[Constructor(optional BodyInit? body = null, optional ResponseInit init), Exposed=(Window,Worker)]
interface Response {
  [NewObject] static Response error();
  [NewObject] static Response redirect(USVString url, optional unsigned short status = 302);

  readonly attribute ResponseType type;

  readonly attribute USVString url;
  readonly attribute boolean redirected;
  readonly attribute unsigned short status;
  readonly attribute boolean ok;
  readonly attribute ByteString statusText;
  [SameObject] readonly attribute Headers headers;
  readonly attribute Promise<Headers> trailer;

  [NewObject] Response clone();
};
Response includes Body;

dictionary ResponseInit {
  unsigned short status = 200;
  ByteString statusText = "";
  HeadersInit headers;
};

enum ResponseType { "basic", "cors", "default", "error", "opaque", "opaqueredirect" };
partial interface mixin WindowOrWorkerGlobalScope {
  [NewObject] Promise<Response> fetch(RequestInfo input, optional RequestInit init);
};
```

File an issue about the selected text