Cache-Control header

- Syntax
  - Cache directives
- Vocabulary
- Directives
  - Response Directives
    - max-age
    - s-maxage
    - no-cache
    - must-revalidate
    - proxy-revalidate
    - no-store
    - private
    - public
    - must-understand
    - no-transform
    - immutable
    - stale-while-revalidate
    - stale-if-error
  - Request Directives
    - no-cache
    - no-store
    - max-age
    - max-stale
    - min-fresh
    - no-transform
    - only-if-cached
    - stale-if-error
- Use Cases

  - Preventing storing
  - Caching static assets with "cache busting"
  - Up-to-date contents always
  - Clearing an already-stored cache
- Specifications
- Browser compatibility
- See also

The HTTP `Cache-Control` header holds *directives* (instructions) in both requests and responses that control caching in browsers and shared caches (e.g., Proxies, CDNs).

Header type	{{Glossary("Request header")}}, {{Glossary("Response header")}}
{{Glossary("Forbidden request header")}}	No
{{Glossary("CORS-safelisted response header")}}	Yes

Syntax

```
Cache-Control: <directive>, <directive>, ...
```

Cache directives follow these rules:

- Caching directives are case-insensitive. However, lowercase is recommended because some implementations do not recognize uppercase directives.
- Multiple directives are permitted and must be comma-separated (e.g., `Cache-control: max-age=180, public`).
- Some directives have an optional argument. When an argument is provided, it is separated from the directive name by an equals symbol (`=`). Typically, arguments for the directives are integers and are therefore not enclosed in quote characters (e.g., `Cache-control: max-age=12`).

Cache directives

The following table lists the standard `Cache-Control` directives:

Request	**Response**
`max-age`	`max-age`

Request	Response
`max-stale`	-
`min-fresh`	-
-	`s-maxage`
`no-cache`	`no-cache`
`no-store`	`no-store`
`no-transform`	`no-transform`
`only-if-cached`	-
-	`must-revalidate`
-	`proxy-revalidate`
-	`must-understand`
-	`private`
-	`public`
-	`immutable`
-	`stale-while-revalidate`
`stale-if-error`	`stale-if-error`

Note: Check the compatibility table for their support; user agents that don't recognize them should ignore them.

Vocabulary

This section defines the terms used in this document, some of which are from the specification.

- (HTTP) cache
  - : Implementation that holds requests and responses for reusing in subsequent requests. It can be either a shared cache or a private cache.
- Shared cache
  - : Cache that exists between the origin server and clients (e.g., Proxy, CDN). It stores a single response and reuses it with multiple users — so developers should avoid storing personalized contents to be cached in the shared cache.
- Private cache
  - : Cache that exists in the client. It is also called *local cache* or *browser cache*. It can store and reuse personalized content for a single user.
- Store response
  - : Store a response in caches when the response is cacheable. However, the cached response is not always reused as-is. (Usually, "cache" means storing a response.)

- Reuse response
  - : Reuse cached responses for subsequent requests.
- Revalidate response
  - : Ask the origin server whether or not the stored response is still fresh. Usually, the revalidation is done through a conditional request.
- Fresh response
  - : Indicates that the response is fresh. This usually means the response can be reused for subsequent requests, depending on request directives.
- Stale response
  - : Indicates that the response is a stale response. This usually means the response can't be reused as-is. Cache storage isn't required to remove stale responses immediately because revalidation could change the response from being stale to being fresh again.
- Age
  - : The time since a response was generated. It is a criterion for whether a response is fresh or stale.

Directives

This section lists directives that affect caching — both response directives and request directives.

Response Directives

`max-age`

The `max-age=N` response directive indicates that the response remains fresh until *N* seconds after the response is generated.

```
Cache-Control: max-age=604800
```

Indicates that caches can store this response and reuse it for subsequent requests while it's fresh.

Note that `max-age` is not the elapsed time since the response was received; it is the elapsed time since the response was generated on the origin server. So if the other cache(s) — on the network route taken by the response — store the response for 100 seconds (indicated using the `Age` response header field), the browser cache would deduct 100 seconds from its freshness lifetime.

If the `max-age` value is negative (for example, `-1`) or isn't an integer (for example, `3599.99`), then the caching behavior is unspecified. Caches are encouraged to treat the value as if it were `0` (this

is noted in the Calculating Freshness Lifetime section of the HTTP specification).

```
Cache-Control: max-age=604800
Age: 100
```

s-maxage

The `s-maxage` response directive indicates how long the response remains fresh in a shared cache. The `s-maxage` directive is ignored by private caches, and overrides the value specified by the `max-age` directive or the `Expires` header for shared caches, if they are present.

```
Cache-Control: s-maxage=604800
```

no-cache

The `no-cache` response directive indicates that the response can be stored in caches, but the response must be validated with the origin server before each reuse, even when the cache is disconnected from the origin server.

```
Cache-Control: no-cache
```

If you want caches to always check for content updates while reusing stored content, `no-cache` is the directive to use. It does this by requiring caches to revalidate each request with the origin server.

Note that `no-cache` does not mean "don't cache". `no-cache` allows caches to store a response but requires them to revalidate it before reuse. If the sense of "don't cache" that you want is actually "don't store", then `no-store` is the directive to use.

> [!NOTE] The `no-cache` directive does not guarantee revalidation for history navigations — such as those made using the Back button. If the back/forward cache ({{Glossary('bfcache')}}) is used, the browser restores a snapshot of the page without revalidating. Even when bfcache is not used, the browser may still serve the cached response without revalidating. This is allowed by the specification because history navigations are usually treated as restoring a snapshot of a historical session and not a new request for a previously visited page.

must-revalidate

The `must-revalidate` response directive indicates that the response can be stored in caches and can be reused while fresh. If the response becomes stale, it must be validated with the origin server before reuse.

Typically, `must-revalidate` is used with `max-age`.

```
Cache-Control: max-age=604800, must-revalidate
```

HTTP allows caches to reuse stale responses when they are disconnected from the origin server. `must-revalidate` is a way to prevent this from happening - either the stored response is revalidated with the origin server or a 504 (Gateway Timeout) response is generated.

> [!NOTE] The `must-revalidate` directive does not guarantee revalidation for history navigations — such as those made using the Back button. If the back/forward cache ({{Glossary('bfcache')}}) is used, the browser restores a snapshot of the page without revalidating. Even when bfcache is not used, the browser may still serve the cached response without revalidating. This is allowed by the specification because history navigations are usually treated as restoring a snapshot of a historical session and not a new request for a previously visited page.

`proxy-revalidate`

The `proxy-revalidate` response directive is the equivalent of `must-revalidate`, but specifically for shared caches only.

`no-store`

The `no-store` response directive indicates that any caches of any kind (private or shared) should not store this response.

```
Cache-Control: no-store
```

`private`

The `private` response directive indicates that the response can be stored only in a private cache (e.g., local caches in browsers).

```
Cache-Control: private
```

You should add the `private` directive for user-personalized content, especially for responses received after login and for sessions managed via cookies.

If you forget to add `private` to a response with personalized content, then that response can be stored in a shared cache and end up being reused for multiple users, which can cause personal information to leak.

`public`

The `public` response directive indicates that the response can be stored in a shared cache. Responses for requests with `Authorization` header fields must not be stored in a shared cache; however, the `public` directive will cause such responses to be stored in a shared cache.

```
Cache-Control: public
```

In general, when pages are under Basic Auth or Digest Auth, the browser sends requests with the `Authorization` header. This means that the response is access-controlled for restricted users (who have accounts), and it's fundamentally not shared-cacheable, even if it has `max-age`.

You can use the `public` directive to unlock that restriction.

```
Cache-Control: public, max-age=604800
```

Note that `s-maxage` or `must-revalidate` also unlock that restriction.

If a request doesn't have an `Authorization` header, or you are already using `s-maxage` or `must-revalidate` in the response, then you don't need to use `public`.

`must-understand`

The `must-understand` response directive indicates that a cache should store the response only if it understands the requirements for caching based on status code.

`must-understand` should be coupled with `no-store` for fallback behavior.

```
Cache-Control: must-understand, no-store
```

If a cache doesn't support `must-understand`, it will be ignored. If `no-store` is also present, the response isn't stored.

If a cache supports `must-understand`, it stores the response with an understanding of cache requirements based on its status code.

`no-transform`

Some intermediaries transform content for various reasons. For example, some convert images to reduce transfer size. In some cases, this is undesirable for the content provider.

`no-transform` indicates that any intermediary (regardless of whether it implements a cache) shouldn't transform the response contents.

`immutable`

The `immutable` response directive indicates that the response will not be updated while it's fresh.

```
Cache-Control: public, max-age=604800, immutable
```

A modern best practice for static resources is to include version/hashes in their URLs, while never modifying the resources — but instead, when necessary, *updating* the resources with newer versions that have new version-numbers/hashes, so that their URLs are different. That's called the **cache-busting** pattern.

```
<script src="https://example.com/react.0.0.0.js"></script>
```

When a user reloads the browser, the browser will send conditional requests for validating to the origin server. But it's not necessary to revalidate those kinds of static resources even when a user reloads the browser, because they're never modified. `immutable` tells a cache that the response is immutable while it's fresh and avoids those kinds of unnecessary conditional requests to the server.

When you use a cache-busting pattern for resources and apply them to a long `max-age`, you can also add `immutable` to avoid revalidation.

`stale-while-revalidate`

The `stale-while-revalidate` response directive indicates that the cache could reuse a stale response while it revalidates it to a cache.

```
Cache-Control: max-age=604800, stale-while-revalidate=86400
```

In the example above, the response is fresh for 7 days (604800s). After 7 days it becomes stale, but the cache is allowed to reuse it for any requests that are made in the following day (86400s), provided that they revalidate the response in the background.

Revalidation will make the cache be fresh again, so it appears to clients that it was always fresh during that period — effectively hiding the latency penalty of revalidation from them.

If no request happened during that period, the cache became stale and the next request will revalidate normally.

`stale-if-error`

The `stale-if-error` response directive indicates that the cache can reuse a stale response when an upstream server generates an error, or when the error is generated locally. Here, an error is considered any response with a status code of 500, 502, 503, or 504.

```
Cache-Control: max-age=604800, stale-if-error=86400
```

In the example above, the response is fresh for 7 days (604800s). Afterwards, it becomes stale, but can be used for an extra 1 day (86400s) when an error is encountered.

After the stale-if-error period passes, the client will receive any error generated.

Request Directives

`no-cache`

The `no-cache` request directive asks caches to validate the response with the origin server before reuse.

```
Cache-Control: no-cache
```

`no-cache` allows clients to request the most up-to-date response even if the cache has a fresh response.

Browsers usually add `no-cache` to requests when users are **force reloading** a page.

`no-store`

The `no-store` request directive allows a client to request that caches refrain from storing the request and corresponding response — even if the origin server's response could be stored.

```
Cache-Control: no-store
```

`max-age`

The `max-age=N` request directive indicates that the client allows a stored response that is generated on the origin server within *N* seconds — where *N* may be any non-negative integer (including `0`).

```
Cache-Control: max-age=10800
```

In the case above, if the response with `Cache-Control: max-age=10800` was generated more than 3 hours ago (calculated from `max-age` and the `Age` header), the cache couldn't reuse that response.

Many browsers use this directive for **reloading**, as explained below.

```
Cache-Control: max-age=0
```

`max-age=0` is a workaround for `no-cache`, because many old (HTTP/1.0) cache implementations don't support `no-cache`. Recently browsers are still using `max-age=0` in "reloading" — for backward compatibility — and alternatively using `no-cache` to cause a "force reloading".

If the `max-age` value is negative (for example, `-1`) or isn't an integer (for example, `3599.99`), then the caching behavior is unspecified. Caches are encouraged to treat the value as if it were `0`.

> [!NOTE] The `max-age` directive does not guarantee revalidation for history navigations — such as those made using the `Back` button. If the back/forward cache ({{Glossary('bfcache')}}) is used, the browser restores a snapshot of the page without revalidating. Even when bfcache is not used, the browser may still serve the cached response without revalidating. This is allowed by the specification because history navigations are usually treated as restoring a snapshot of a historical session and not a new request for a previously visited page.

`max-stale`

The `max-stale=N` request directive indicates that the client allows a stored response that is stale within *N* seconds. If no *N* value is specified, the client will accept a stale response of any age.

```
Cache-Control: max-stale=3600
```

For example, a request with the header above indicates that the browser will accept a stale response from the cache that has expired within the last hour.

Clients can use this header when the origin server is down or too slow and can accept cached responses from caches even if they are a bit old.

Note that the major browsers do not support requests with `max-stale`.

`min-fresh`

The `min-fresh=N` request directive indicates that the client allows a stored response that is fresh for at least *N* seconds.

```
Cache-Control: min-fresh=600
```

In the case above, if the response with `Cache-Control: max-age=3600` was stored in caches 51 minutes ago, the cache couldn't reuse that response.

Clients can use this header when the user requires the response to not only be fresh, but also requires that it won't be updated for a period of time.

Note that the major browsers do not support requests with `min-fresh`.

`no-transform`

Same meaning that `no-transform` has for a response, but for a request instead.

`only-if-cached`

The client indicates that an already-cached response should be returned. If a cache has a stored response, even a stale one, it will be returned. If no cached response is available, a 504 Gateway Timeout response will be returned.

`stale-if-error`

The `stale-if-error` request directive indicates that the browser is interested in receiving stale content on error from any intermediate server for a particular origin. This is not supported by any browser (see Browser compatibility).

Use Cases

Preventing storing

If you don't want a response stored in caches, use the `no-store` directive.

```
Cache-Control: no-store
```

Note that `no-cache` means "it can be stored but don't reuse before validating" — so it's not for preventing a response from being stored.

```
Cache-Control: no-cache
```

In theory, if directives are conflicted, the most restrictive directive should be honored. So the example below is basically meaningless because `private`, `no-cache`, `max-age=0` and `must-revalidate` conflict with `no-store`.

```
# conflicted
Cache-Control: private, no-cache, no-store, max-age=0, must-revalidate
```

```
# equivalent to
Cache-Control: no-store
```

Caching static assets with "cache busting"

When you build static assets with versioning/hashing mechanisms, adding a version/hash to the filename or query string is a good way to manage caching.

For example:

```
<!-- index.html -->
<script src="/assets/react.min.js"></script>
<img src="/assets/hero.png" width="900" height="400" />
```

The React library version will change when you update the library, and `hero.png` will also change when you edit the picture. So those are hard to store in a cache with `max-age`.

In such a case, you could address the caching needs by using a specific, numbered version of the library, and including the hash of the picture in its URL.

```
<!-- index.html -->
<script src="/assets/react.0.0.0min.js"></script>
<img src="/assets/hero.png?hash=deadbeef" width="900" height="400" />
```

You can add a long `max-age` value and `immutable` because the content will never change.

```
# /assets/*
Cache-Control: max-age=31536000, immutable
```

When you update the library or edit the picture, new content should have a new URL, and caches aren't reused. That is called the "cache busting" pattern.

Use a `no-cache` to make sure that the HTML response itself is not cached. `no-cache` could cause revalidation, and the client will correctly receive a new version of the HTML response and static assets.

```
# /index.html
Cache-Control: no-cache
```

Note: If `index.html` is controlled under Basic Authentication or Digest Authentication, files under `/assets` are not stored in the shared cache. If `/assets/` files are suitable for storing in a shared

cache, you also need one of `public`, `s-maxage` or `must-revalidate`.

Up-to-date contents always

For content that's generated dynamically, or that's static but updated often, you want a user to always receive the most up-to-date version.

If you don't add a `Cache-Control` header because the response is not intended to be cached, that could cause an unexpected result. Cache storage is allowed to cache it heuristically — so if you have any requirements on caching, you should always indicate them explicitly, in the `Cache-Control` header.

Adding `no-cache` to the response causes revalidation to the server, so you can serve a fresh response every time — or if the client already has a new one, just respond `304 Not Modified`.

```
Cache-Control: no-cache
```

Most HTTP/1.0 caches don't support `no-cache` directives, so historically `max-age=0` was used as a workaround. But only `max-age=0` could cause a stale response to be reused when caches disconnected from the origin server. `must-revalidate` addresses that. That's why the example below is equivalent to `no-cache`.

```
Cache-Control: max-age=0, must-revalidate
```

But for now, you can simply use `no-cache` instead.

Clearing an already-stored cache

There are no cache directives for clearing already-stored responses from caches on *intermediate* servers.

Imagine that clients/caches store a fresh response for a path, with no request flight to the server. There is nothing a server could do to that path.

`Clear-Site-Data: cache` can be used to clear every stored response for a site in the browser cache, so use this with care. Note that this will not affect shared or intermediate caches.

Specifications

{{Specifications}}

Browser compatibility

{{Compat}}

See also

- HTTP caching
- Caching Tutorial for Web Authors and Webmasters
- Caching best practices & max-age gotchas
- Cache-Control for Civilians
- RFC 9111 – HTTP Caching
- RFC 5861 – HTTP Cache-Control Extensions for Stale Content
- RFC 8246 – HTTP Immutable Responses

□□□ #2
□ □□□ □□□ 2026-10-31 03:27:53 UTC
□ □□□ □□□ 2026-10-31 03:27:53 UTC