part of Hypertext Transfer Protocol -- HTTP/1.1
RFC 2616 Fielding, et al.

14 Header Field Definitions

This section defines the syntax and semantics of all standard HTTP/1.1 header fields. For entity-header fields, both sender and recipient refer to either the client or the server, depending on who sends and who receives the entity.

14.1 Accept

The Accept request-header field can be used to specify certain media types which are acceptable for the response. Accept headers can be used to indicate that the request is specifically limited to a small set of desired types, as in the case of a request for an in-line image.

```
Accept         = "Accept" ":"
                 #( media-range [ accept-params ] )

media-range    = ( "*/*"
                 | ( type "/" "*" )
                 | ( type "/" subtype )
                 ) *( ";" parameter )
accept-params  = ";" "q" "=" qvalue *( accept-extension )
accept-extension = ";" token [ "=" ( token | quoted-string ) ]
```

The asterisk "*" character is used to group media types into ranges, with "*/*" indicating all media types and "type/*" indicating all subtypes of that type. The media-range MAY include media type parameters that are applicable to that range.

Each media-range MAY be followed by one or more accept-params, beginning with the "q" parameter for indicating a relative quality factor. The first "q" parameter (if any) separates the media-range parameter(s) from the accept-params. Quality factors allow the user or user agent to indicate the relative degree of preference for that media-range, using the qvalue scale from 0 to 1 (section 3.9). The default value is q=1.

```
Note: Use of the "q" parameter name to separate media type
parameters from Accept extension parameters is due to historical
practice. Although this prevents any media type parameter named
"q" from being used with a media range, such an event is believed
to be unlikely given the lack of any "q" parameters in the IANA
media type registry and the rare usage of any media type
parameters in Accept. Future media types are discouraged from
registering any parameter named "q".
```

The example

```
Accept: audio/*; q=0.2, audio/basic
```

SHOULD be interpreted as "I prefer audio/basic, but send me any audio type if it is the best available after an 80% mark-down in quality."

If no Accept header field is present, then it is assumed that the client accepts all media types. If an Accept header field is present, and if the server cannot send a response which is acceptable according to the combined Accept field value, then the server SHOULD send a 406 (not acceptable) response.

A more elaborate example is

```
Accept: text/plain; q=0.5, text/html,
        text/x-dvi; q=0.8, text/x-c
```

Verbally, this would be interpreted as "text/html and text/x-c are the preferred media types, but if they do not exist, then send the text/x-dvi entity, and if that does not exist, send the text/plain entity."

Media ranges can be overridden by more specific media ranges or specific media types. If more than one media range applies to a given type, the most specific reference has precedence. For example,

```
Accept: text/*, text/html, text/html;level=1, */*
```

have the following precedence:

```
1) text/html;level=1
2) text/html
3) text/*
4) */*
```

The media type quality factor associated with a given type is determined by finding the media range with the highest precedence which matches that type. For example,

```
Accept: text/*;q=0.3, text/html;q=0.7, text/html;level=1,
        text/html;level=2;q=0.4, */*;q=0.5
```

would cause the following values to be associated:

```
text/html;level=1         = 1
text/html                 = 0.7
text/plain                = 0.3

image/jpeg                = 0.5
text/html;level=2         = 0.4
text/html;level=3         = 0.7
```

```
Note: A user agent might be provided with a default set of quality
values for certain media ranges. However, unless the user agent is
a closed system which cannot interact with other rendering agents,
this default set ought to be configurable by the user.
```

14.2 Accept-Charset

The Accept-Charset request-header field can be used to indicate what character sets are acceptable for the response. This field allows clients capable of understanding more comprehensive or special- purpose character sets to signal that capability to a server which is capable of representing documents in those character sets.

```
Accept-Charset = "Accept-Charset" ":"
        1#( ( charset | "*" )[ ";" "q" "=" qvalue ] )
```

Character set values are described in section 3.4. Each charset MAY be given an associated quality value which represents the user's preference for that charset. The default value is q=1. An example is

```
Accept-Charset: iso-8859-5, unicode-1-1;q=0.8
```

The special value "*", if present in the Accept-Charset field, matches every character set (including ISO-8859-1) which is not mentioned elsewhere in the Accept-Charset field. If no "*" is present in an Accept-Charset field, then all character sets not explicitly mentioned get a quality value of 0, except for ISO-8859-1, which gets a quality value of 1 if not explicitly mentioned.

If no Accept-Charset header is present, the default is that any character set is acceptable. If an Accept-Charset header is present, and if the server cannot send a response which is acceptable according to the Accept-Charset header, then the server SHOULD send an error response with the 406 (not acceptable) status code, though the sending of an unacceptable response is also allowed.

14.3 Accept-Encoding

The Accept-Encoding request-header field is similar to Accept, but restricts the content-codings (section 3.5) that are acceptable in the response.

```
Accept-Encoding  = "Accept-Encoding" ":"

                   1#( codings [ ";" "q" "=" qvalue ] )
codings          = ( content-coding | "*" )
```

Examples of its use are:

```
Accept-Encoding: compress, gzip
Accept-Encoding:
Accept-Encoding: *
Accept-Encoding: compress;q=0.5, gzip;q=1.0
Accept-Encoding: gzip;q=1.0, identity; q=0.5, *;q=0
```

A server tests whether a content-coding is acceptable, according to an Accept-Encoding field, using these rules:

1. If the content-coding is one of the content-codings listed in the Accept-Encoding field, then it is acceptable, unless it is accompanied by a qvalue of 0. (As defined in section 3.9, a qvalue of 0 means "not acceptable.")

2. The special "*" symbol in an Accept-Encoding field matches any available content-coding not explicitly listed in the header field.

3. If multiple content-codings are acceptable, then the acceptable content-coding with the highest non-zero qvalue is preferred.

4. The "identity" content-coding is always acceptable, unless specifically refused because the Accept-Encoding field includes "identity;q=0", or because the field includes "*;q=0" and does not explicitly include the "identity" content-coding. If the Accept-Encoding field-value is empty, then only the "identity" encoding is acceptable.

If an Accept-Encoding field is present in a request, and if the server cannot send a response which is acceptable according to the Accept-Encoding header, then the server SHOULD send an error response with the 406 (Not Acceptable) status code.

If no Accept-Encoding field is present in a request, the server MAY assume that the client will accept any content coding. In this case, if "identity" is one of the available content-codings, then the server SHOULD use the "identity" content-coding, unless it has additional information that a different content-coding is meaningful to the client.

```
Note: If the request does not include an Accept-Encoding field,
and if the "identity" content-coding is unavailable, then
content-codings commonly understood by HTTP/1.0 clients (i.e.,

"gzip" and "compress") are preferred; some older clients
improperly display messages sent with other content-codings.  The
server might also make this decision based on information about
the particular user-agent or client.

Note: Most HTTP/1.0 applications do not recognize or obey qvalues
associated with content-codings. This means that qvalues will not
work and are not permitted with x-gzip or x-compress.
```

14.4 Accept-Language

The Accept-Language request-header field is similar to Accept, but restricts the set of natural languages that are preferred as a response to the request. Language tags are defined in section 3.10.

```
Accept-Language = "Accept-Language" ":"
                  1#( language-range [ ";" "q" "=" qvalue ] )
language-range  = ( ( 1*8ALPHA *( "-" 1*8ALPHA ) ) | "*" )
```

Each language-range MAY be given an associated quality value which represents an estimate of the user's preference for the languages specified by that range. The quality value defaults to "q=1". For example,

```
Accept-Language: da, en-gb;q=0.8, en;q=0.7
```

would mean: "I prefer Danish, but will accept British English and other types of English." A language-range matches a language-tag if it exactly equals the tag, or if it exactly equals a prefix of the tag such that the first tag character following the prefix is "-". The special range "*", if present in the Accept-Language field, matches every tag not matched by any other range present in the Accept-Language field.

```
Note: This use of a prefix matching rule does not imply that
language tags are assigned to languages in such a way that it is
always true that if a user understands a language with a certain
tag, then this user will also understand all languages with tags
for which this tag is a prefix. The prefix rule simply allows the
use of prefix tags if this is the case.
```

The language quality factor assigned to a language-tag by the Accept-Language field is the quality value of the longest language- range in the field that matches the language-tag. If no language- range in the field matches the tag, the language quality factor assigned is 0. If no Accept-Language header is present in the request, the server

SHOULD assume that all languages are equally acceptable. If an Accept-Language header is present, then all languages which are assigned a quality factor greater than 0 are acceptable.

It might be contrary to the privacy expectations of the user to send an Accept-Language header with the complete linguistic preferences of the user in every request. For a discussion of this issue, see section 15.1.4.

As intelligibility is highly dependent on the individual user, it is recommended that client applications make the choice of linguistic preference available to the user. If the choice is not made available, then the Accept-Language header field MUST NOT be given in the request.

```
Note: When making the choice of linguistic preference available to
the user, we remind implementors of  the fact that users are not
familiar with the details of language matching as described above,
and should provide appropriate guidance. As an example, users
might assume that on selecting "en-gb", they will be served any
kind of English document if British English is not available. A
user agent might suggest in such a case to add "en" to get the
best matching behavior.
```

14.5 Accept-Ranges

```
The Accept-Ranges response-header field allows the server to
indicate its acceptance of range requests for a resource:

    Accept-Ranges     = "Accept-Ranges" ":" acceptable-ranges
    acceptable-ranges = 1#range-unit | "none"

Origin servers that accept byte-range requests MAY send

    Accept-Ranges: bytes

but are not required to do so. Clients MAY generate byte-range
requests without having received this header for the resource
```

involved. Range units are defined in section 3.12.

Servers that do not accept any kind of range request for a resource MAY send

```
Accept-Ranges: none
```

to advise the client not to attempt a range request.

14.6 Age

The Age response-header field conveys the sender's estimate of the amount of time since the response (or its revalidation) was generated at the origin server. A cached response is "fresh" if its age does not exceed its freshness lifetime. Age values are calculated as specified in section 13.2.3.

```
Age = "Age" ":" age-value
age-value = delta-seconds
```

Age values are non-negative decimal integers, representing time in seconds.

If a cache receives a value larger than the largest positive integer it can represent, or if any of its age calculations overflows, it MUST transmit an Age header with a value of 2147483648 (2^31). An HTTP/1.1 server that includes a cache MUST include an Age header field in every response generated from its own cache. Caches SHOULD use an arithmetic type of at least 31 bits of range.

14.7 Allow

The Allow entity-header field lists the set of methods supported by the resource identified by the Request-URI. The purpose of this field is strictly to inform the recipient of valid methods associated with the resource. An Allow header field MUST be present in a 405 (Method Not Allowed) response.

```
Allow   = "Allow" ":" #Method
```

Example of use:

```
Allow: GET, HEAD, PUT
```

This field cannot prevent a client from trying other methods. However, the indications given by the Allow header field value SHOULD be followed. The actual set of allowed methods is defined by the origin server at the time of each request.

The Allow header field MAY be provided with a PUT request to recommend the methods to be supported by the new or modified resource. The server is not required to support these methods and SHOULD include an Allow header in the response giving the actual supported methods.

A proxy MUST NOT modify the Allow header field even if it does not understand all the methods specified, since the user agent might have other means of communicating with the origin server.

14.8 Authorization

A user agent that wishes to authenticate itself with a server--usually, but not necessarily, after receiving a 401 response--does so by including an Authorization request-header field with the request. The Authorization field value consists of credentials containing the authentication information of the user agent for the realm of the resource being requested.

```
    Authorization  = "Authorization" ":" credentials
```

HTTP access authentication is described in "HTTP Authentication: Basic and Digest Access Authentication" [43]. If a request is authenticated and a realm specified, the same credentials SHOULD be valid for all other requests within this realm (assuming that the authentication scheme itself does not require otherwise, such as credentials that vary according to a challenge value or using synchronized clocks).

When a shared cache (see section 13.7) receives a request containing an Authorization field, it MUST NOT return the corresponding response as a reply to any other request, unless one of the following specific exceptions holds:

1. If the response includes the "s-maxage" cache-control directive, the cache MAY use that response in replying to a subsequent request. But (if the specified maximum age has passed) a proxy cache MUST first revalidate it with the origin server, using the request-headers from the new request to allow the origin server to authenticate the new request. (This is the defined behavior for s-maxage.) If the response includes "s-maxage=0", the proxy MUST always revalidate it before re-using it.

2. If the response includes the "must-revalidate" cache-control directive, the cache MAY use that response in replying to a subsequent request. But if the response is stale, all caches MUST first revalidate it with the origin server, using the request-headers from the new request to allow the origin server to authenticate the new request.

3. If the response includes the "public" cache-control directive, it MAY be returned in reply to any subsequent request.

14.9 Cache-Control

The Cache-Control general-header field is used to specify directives that MUST be obeyed by all caching mechanisms along the request/response chain. The directives specify behavior intended to prevent caches from adversely interfering with the request or response. These directives typically override the default caching algorithms. Cache directives are unidirectional in that the presence of a directive in a request does not imply that the same directive is to be given in the response.

Note that HTTP/1.0 caches might not implement Cache-Control and might only implement Pragma: no-cache (see section 14.32).

Cache directives MUST be passed through by a proxy or gateway application, regardless of their significance to that application, since the directives might be applicable to all recipients along the request/response chain. It is not possible to specify a cache- directive for a specific cache.

```
Cache-Control   = "Cache-Control" ":" 1#cache-directive

cache-directive = cache-request-directive
     | cache-response-directive

cache-request-directive =
       "no-cache"                          ; Section 14.9.1
     | "no-store"                          ; Section 14.9.2
     | "max-age" "=" delta-seconds         ; Section 14.9.3, 14.9.4
     | "max-stale" [ "=" delta-seconds ]   ; Section 14.9.3
     | "min-fresh" "=" delta-seconds       ; Section 14.9.3
     | "no-transform"                      ; Section 14.9.5
     | "only-if-cached"                    ; Section 14.9.4
     | cache-extension                     ; Section 14.9.6

 cache-response-directive =
       "public"                               ; Section 14.9.1
```

```
        | "private" [ "=" <"> 1#field-name <"> ] ; Section 14.9.1
        | "no-cache" [ "=" <"> 1#field-name <"> ]; Section 14.9.1
        | "no-store"                             ; Section 14.9.2
        | "no-transform"                         ; Section 14.9.5
        | "must-revalidate"                      ; Section 14.9.4
        | "proxy-revalidate"                     ; Section 14.9.4
        | "max-age" "=" delta-seconds            ; Section 14.9.3
        | "s-maxage" "=" delta-seconds           ; Section 14.9.3
        | cache-extension                        ; Section 14.9.6

    cache-extension = token [ "=" ( token | quoted-string ) ]
```

When a directive appears without any 1#field-name parameter, the directive applies to the entire request or response. When such a directive appears with a 1#field-name parameter, it applies only to the named field or fields, and not to the rest of the request or response. This mechanism supports extensibility; implementations of future versions of the HTTP protocol might apply these directives to header fields not defined in HTTP/1.1.

The cache-control directives can be broken down into these general categories:

```
- Restrictions on what are cacheable; these may only be imposed by
  the origin server.

- Restrictions on what may be stored by a cache; these may be
  imposed by either the origin server or the user agent.

- Modifications of the basic expiration mechanism; these may be
  imposed by either the origin server or the user agent.

- Controls over cache revalidation and reload; these may only be
  imposed by a user agent.

- Control over transformation of entities.

- Extensions to the caching system.
```


14.9.1 What is Cacheable

By default, a response is cacheable if the requirements of the request method, request header fields, and the response status indicate that it is cacheable. Section 13.4 summarizes these defaults for cacheability. The following Cache-Control response directives allow an origin server to override the default cacheability of a response:

public
: Indicates that the response MAY be cached by any cache, even if it would normally be non-cacheable or cacheable only within a non- shared cache. (See also Authorization, section 14.8, for additional details.)

private
: Indicates that all or part of the response message is intended for a single user and MUST NOT be cached by a shared cache. This allows an origin server to state that the specified parts of the response are intended for only one user and are not a valid response for requests by other users. A private (non-shared) cache MAY cache the response.

  Note: This usage of the word private only controls where the response may be cached, and cannot ensure the privacy of the message content.

no-cache
: If the no-cache directive does not specify a field-name, then a cache MUST NOT use the response to satisfy a subsequent request without successful revalidation with the origin server. This allows an origin server to prevent caching even by caches that have been configured to return stale responses to client requests.

If the no-cache directive does specify one or more field-names, then a cache MAY use the response to satisfy a subsequent request, subject to any other restrictions on caching. However, the specified field-name(s) MUST NOT be sent in the response to a subsequent request without successful revalidation with the origin server. This allows an origin server to prevent the re-use of certain header fields in a response, while still allowing caching of the rest of the response.

Note: Most HTTP/1.0 caches will not recognize or obey this directive.

14.9.2 What May be Stored by Caches

no-store

The purpose of the no-store directive is to prevent the inadvertent release or retention of sensitive information (for example, on backup tapes). The no-store directive applies to the entire message, and MAY be sent either in a response or in a request. If sent in a request, a cache MUST NOT store any part of either this request or any response to it. If sent in a response, a cache MUST NOT store any part of either this response or the request that elicited it. This directive applies to both non-shared and shared caches. "MUST NOT store" in this context means that the cache MUST NOT intentionally store the information in non-volatile storage, and MUST make a best-effort attempt to remove the information from volatile storage as promptly as possible after forwarding it.

Even when this directive is associated with a response, users might explicitly store such a response outside of the caching system (e.g., with a "Save As" dialog). History buffers MAY store such responses as part of their normal operation.

The purpose of this directive is to meet the stated requirements of certain users and service authors who are concerned about accidental releases of information via unanticipated accesses to cache data structures. While the use of this directive might improve privacy in some cases, we caution that it is NOT in any way a reliable or sufficient mechanism for ensuring privacy. In particular, malicious or compromised caches might not recognize or obey this directive, and communications networks might be vulnerable to eavesdropping.

14.9.3 Modifications of the Basic Expiration Mechanism

The expiration time of an entity MAY be specified by the origin server using the Expires header (see section 14.21). Alternatively, it MAY be specified using the max-age directive in a response. When the max-age cache-control directive is present in a cached response, the response is stale if its current age is greater than the age value given (in seconds) at the time of a new request for that resource. The max-age directive on a response implies that the response is cacheable (i.e., "public") unless some other, more restrictive cache directive is also present.

If a response includes both an Expires header and a max-age directive, the max-age directive overrides the Expires header, even if the Expires header is more restrictive. This rule allows an origin server to provide, for a given response, a longer expiration time to an HTTP/1.1 (or later) cache than to an HTTP/1.0 cache. This might be useful if certain HTTP/1.0 caches improperly calculate ages or expiration times, perhaps due to desynchronized clocks.

Many HTTP/1.0 cache implementations will treat an Expires value that is less than or equal to the response Date value as being equivalent to the Cache-Control response directive "no-cache". If an HTTP/1.1 cache receives such a response, and the response does not include a Cache-Control header field, it SHOULD consider the response to be non-cacheable in order to retain compatibility with HTTP/1.0 servers.

Note: An origin server might wish to use a relatively new HTTP cache control feature, such as the "private" directive, on a network including older caches that do not understand that feature. The origin server will need to combine the new feature with an Expires field whose value is less than or equal to the Date value. This will prevent older caches from improperly caching the response.

s-maxage
: If a response includes an s-maxage directive, then for a shared cache (but not for a private cache), the maximum age specified by this directive overrides the maximum age specified by either the max-age directive or the Expires header. The s-maxage directive also implies the semantics of the proxy-revalidate directive (see section 14.9.4), i.e., that the shared cache must not use the entry after it becomes stale to respond to a subsequent request without first revalidating it with the origin server. The s- maxage directive is always ignored by a private cache.

Note that most older caches, not compliant with this specification, do not implement any cache-control directives. An origin server wishing to use a cache-control directive that restricts, but does not prevent, caching by an HTTP/1.1-compliant cache MAY exploit the requirement that the max-age directive overrides the Expires header, and the fact that pre-HTTP/1.1-compliant caches do not observe the max-age directive.

Other directives allow a user agent to modify the basic expiration mechanism. These directives MAY be specified on a request:

max-age
: Indicates that the client is willing to accept a response whose age is no greater than the specified time in seconds. Unless max- stale directive is also included, the client is not willing to accept a stale response.

min-fresh
: Indicates that the client is willing to accept a response whose freshness lifetime is no less than its current age plus the specified time in seconds. That is, the client wants a response that will still be fresh for at least the specified number of seconds.