Header fields in HTTP

Header fields provide required information about requests or responses, or about objects sent in the notification body. There are 4 types of HTTP header Header.

Header fields provide required information about requests or responses, or about objects sent in the notification body. There are 4 types of HTTP notification Header:

1. **General-Header** : These Header fields have general application capabilities for both request and response notifications.
2. **Request Type (Request-Header)** : These Header fields are applicable to only required messages.
3. **Response Type (Response-Header)** : These Header fields are only applicable for response messages.
4. **Entity-Header** : These fields define the body-entity information or, if no body is present, about the source identified by the request.

General Header

Cache-Control School

Common Header Field **Cache-Control** is used to identify instructions that **MUST** be followed by all hidden memory systems. The syntax is as follows:

```
Cache-Control: cache-request-directive | cache-response-directive
```

A Client or Server can use the Common Header **Cache-Control** to determine parameters for hidden memory or request specific types of documents from hidden memory. Hidden memory instructions are defined in a list separated by commas. For example:

```
Cache-control: no-cache
```

The table below lists the instructions that require important hidden memory that can be used by the **Client** in its HTTP request:

STT Indicates the need for hidden memory and describes 1 **no-cache**

A hidden memory must not use the response to satisfy a subsequent request without re-confirming success with the original Server.

2 **no-store**

The cache should not store anything about the Client request or the Server counter.

3 **max-age = seconds (s)**

Indicates that the Client wants to accept a response whose time is not greater than the time specified in seconds (s).

4 **max-stale [in seconds]**

Indicates that the Client is willing to accept a response that has exceeded the expiration time. If the number of seconds is provided, it must not be expired by more than that time.

5 **min-fresh = seconds**

Indicates that the Client is willing to accept a response whose health life is no less than its current age plus the specified time in seconds.

6 **no-transform**

Do not convert the body of the object.

7 **only-if-cached**

Do not get new data. Hidden memory can send a document only if it is in hidden memory, and should not contact the original Server to consider if a newer copy exists.

The following important hidden memory response instructions can be used by the Server in its response:

STT Hide memory feedback instructions and Description1 **public**

Indicates that feedback can be kept in memory hidden by any hidden memory.

2 **private**

Indicates that all or part of the response message is considered as for a single user and must not be kept in hidden memory by a shared cache.

3 **no-cache**

A hidden memory must not use feedback to satisfy a subsequent request without re-confirming success with the original Server.

4 **no-store**

Caching should not save anything about Client requests or Server responses.

5 **no-transform**

Do not convert the body of the object.

6 **must-revalidate**

The cache must verify the status of old documents before using it and expired documents should not be used.

7 **proxy-revalidate**

The instruction to re-authorize the authorization has the same meaning as the must-revalidate directive, except that it does not apply to hidden user agent memory that is not shared.

8 **max-age = seconds**

Indicates that the client is willing to accept a request whose age is not greater than the specified time in seconds.

9 **s-maxage = seconds**

The maximum age defined by this directive exceeds the maximum age specified by either the max-age directive or Expires Header. The s-maxage instruction is always ignored by an individual memory.

Connection field

The **Connection** Common **Head field** allows the sender to determine the functions that are expected for that particular connection and must not be communicated by proxy stations over further connections. Here is a simple syntax for using the Connection Header:

```
Connection: "Connection"
```

HTTP / 1.1 specifies the " **close** " connection function for the sender to the signal that the connection will be closed after the response is completed. For example:

```
Connection: close
```

By default, HTTP 1.1 uses persistent connections, where the connection does not automatically close after completing a transaction. Meanwhile, HTTP 1.0 does not have persistent connections by default. If a Client 1.0 wishes to use persistent connections, it uses **keep-alvie parameters** as follows:

```
Connection: keep-alive
```

Date field

All Date / Time labels **MUST** be performed in GMT, with no exceptions. HTTP applications are allowed to use any of the following 3 representations of the Date / Time label:

```
Sun, 06 Nov 1994 08:49:37 GMT;  RFC 822, updated by RFC 1123
Sunday, 06-Nov-94 08:49:37 GMT;  RFC 850, obsoleted by RFC 1036
Sun Nov 6 08:49:37 1994;  ANSI C's asctime () format
```

Here, the first format is the most used.

Pragma school

The Pragma field is used to include specific instructions for implementation that can be applied to any recipient in the Request / Response sequence. For example:

```
Pragma: no-cache
```

The instructions specified only in HTTP / 1.0 are hidden memory instructions and are maintained in HTTP / 1.1 for backward compatibility. There are no new Pragma instructions to be defined in the future.

School Trailer

The **Trailer field** value indicates that the given setting of the Header fields represented in the trailer of an encrypted message with the transport encoding is closed. Here is the syntax of the Trailer school:

```
Trailer: field-name
```

The message Header fields listed in the Trailer field must not include the following Header fields:

1. Transfer-Encoding
2. Content-Length
3. Trailer

Transfer-Encoding field

This Transfer-Encoding field indicates which type of transmission is applied to the message body so that the transmission is safe between the sender and the receiver. This is not the same as **Content-encoding** because the transport encoding is an attribute of the message, not the message body. The syntax of the Transfer-Encoding field is as follows:

```
Transfer-Encoding: chunked
```

All Transfer-Encoding values ??are insensitive (non-case-sensitive).

School Upgrade

This Upgrade field allows the Client to identify additional communication protocols that it supports and will be used if the Server finds that it is suitable for protocol conversion. For example:

```
Upgrade: HTTP / 2.0, SHTTP / 1.3, IRC / 6.9, RTA / x11
```

The Upgrade field is expected to provide a simple technique for transmitting from HTTP / 1.1 to some incompatible protocols.

Via school

The Via field must be used by gateways and proxy stations to indicate intermediate and recipient protocols. For example, a request message can be sent from an HTTP / 1.0 User agent to an internal proxy station named "fred", which uses HTTP / 1.1 to forward the request to a proxy station. Public at nowhere.com, which is completed by forwarding it to the original Server at www.ics.uci.edu. Requests received by www.ics.uci.edu will have the Via school as follows:

```
Via: 1.0 fred, 1.1 nowhere.com (Apache / 1.1)
```

Warning field (Warning)

The Warning field is used to bring more information about the status or transmission of a message that may not be reflected in that message. A response may carry more than a Warning field.

```
Warning: warn-agent SP warn-agent SP warn-text SP warn-date
```

The Header fields required on the Client

Accept (Accept)

This Acceptance can be used to identify specific media types that are acceptable for feedback. The general syntax is as follows:

```
Accept: type / subtype [q = qvalue]
```

Media types can be listed separately by commas and arbitrary q values ??represent an acceptable level of quality to accept types on a range from 0 to 1. Here is For example:

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

This paragraph can be compiled as **text / html** and **text / xc** and is the preferred media type but if they do not exist, then send the **text / x-dvi** object, and if it does not exist , send the **text / plain** object.

Accept-Charset field

This field can be used to indicate which character sets are accepted for feedback. Here is the general syntax:

```
Accept-Charset: character_set [q = qvalue]
```

Many character sets can be listed separately by commas and arbitrary q values ??represent an acceptable level of quality for non-preferred character sets on a domain from 0 to 1 Here is an example:

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

The special value "*", if available in the **Accept-Charset field** , connects all character sets and if there is no value of the **Accept-Charset** field, then by default any set of characters can be accepted. .

Accept-Encoding field

This field is similar to Accept, but content content restriction is acceptable in the response. The general syntax is:

```
Accept-Encoding: encoding types
```

The examples are as follows:

```
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
```

Accept-Language field

This field is similar to Accept, but limiting the set of natural languages ??is preferred when a response to a request. The general syntax is:

```
Accept-Language: language [q = qvalue]
```

Many languages ??can be listed separated by commas and arbitrary q values ??represent an acceptable level of quality for languages ??that are not preferred on the domain from 0 to 1. Here is an example:

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

Authorization field (Authorization)

Authorization field values ??include credentials that contain a user agent's authorization information for the source range being requested. The general syntax is:

```
Authorization: credentials
```

Configure HTTP / 1.0 to define the basic authorization schema, where the proxy parameter is a string of **username:** 64-bit encrypted **password** . Here is an example:

```
Authorization: BASIC Z3Vlc3Q6Z3Vlc3QxMjM =
```

The decoded value is **guest: guest123** , where **guest** is the user account and **guest123** is the password.

Cookie field

Field value **Cookie** contains a name / value pair of information stored for that URL. Here is the general syntax:

```
Cookie: name = value
```

Many cookies can be identified separated by semicolons ";" as follows:

```
Cookie: name1 = value1; name2 = value2; name3 = value3
```

Expect School

The Expect field is used to indicate that a specific set of Server behaviors is required by the Client. The general syntax is:

```
Expect: 100-continue |  expectation-extension
```

If a server receives a request that contains an Expect field that includes an expected extension that it does not support, it must respond to the 417 state (the expected failure).

From school

The **From** field contains an e-mail address for the user whose user agent controls. Here is a simple syntax:

```
From: webmaster@w3.org
```

This field can be used for entering purposes and as a means for confirming the source of requests that are not feasible or not desirable.

Host School

The **Host** field is used to identify the Internet host and the port number of the requested source. The general syntax is:

```
Host: "Host" ":" host [":" port];
```

A **Host** that does not have any port information is assumed to be the default port, which is 80. For example, an initial Server request for http://w3.org/pub/WWW/ will be:

```
GET / pub / WWW / HTTP / 1.1
Host: www.w3.org
```

If-Match field

If-Match field is used in a method to make it conditional. This header requires the Server to represent the requested method only when the value provided in this tag connects to the provided object tags represented by the **Etag** . The general syntax is:

```
If-Match: entity-tag
```

A sign (*) connects to any object, and the transmission continues only when the object exists. Here are possible examples:

```
If-Match: "xyzzy"
If-Match: "xyzzy", "r2d2xxxx", "c3piozzzz"
If-Match: *
```

If no object is connected, or if (*) is provided and no existing object exists, the server not presented method is required, and must return a response of 412 (condition before failure).

If-Modified-Since field

This field is used with a method to make it conditional. If the requested URL is not modified from the time specified in this field, an object will not be returned from the Server; instead, a response of 304 (unmodified) will be returned without any notification body. If-Modified-Since's general syntax is:

```
If-Modified-Since: HTTP-date
```

An example of the school is:

```
If-Modified-Since: Sat, Oct 29, 19:43:31 GMT
```

If no object tag is connected to, or if "*" is provided and no existing object exists, the server is not presented with the required method, and must return response 412 (condition before failure).

If-None-Match field

This field is used with a method to make it conditional. This field requires the server to present the requested method only when one of the given values ??of this tag connects to the object tags provided by the **Etag** . The general syntax is:

```
If-None-Match: entity-tag
```

A sign * connects to any object, and the transmission continues only if the object does not exist. Here are possible examples:

```
If-None-Match: "xyzzy"
If-None-Match: "xyzzy", "r2d2xxxx", "c3piozzzz"
If-None-Match: *
```

If-Range School

The **If-Range field** can be used with a conditional GET to request only a part of the object that is missing, if it is not changed, and the entire object if it is changed. The general syntax is as follows:

```
If-Range: entity-tag |  HTTP-date
```

Either an object tag or a data can be used to verify the received internal object. For example:

```
If-Range: Sat, Oct 29, 19:43:31 GMT
```

Here, if the document is not edited from the given date, the Server returns the byte range provided by the Range field, otherwise it returns all new documents.

If-Unmodified-Since field

This field is used with a method to make it conditional. The general syntax is:

```
If-Unmodified-Since: HTTP-date
```

If the requested source is not modified since the time has been specified in this field, the Server will perform the requested operation if **If-Unmodified-Since is** not performed. For example:

```
If-Unmodified-Since: Sat, Oct 29, 19:43:31 GMT
```

If the request results in anything other than a state of 2xx or 4xx, then the If-Unmodified-Since field should be ignored.

Max-Forwards School

This field provides a technique with TRACE methods and OPTIONS to limit the number of authorized stations or gateways that can forward requests to the next server. This is the general syntax:

```
Max-Forwards: n
```

Max-Forward value is a decimal integer that indicates the remaining number of messages this request can be forwarded. This is useful for debugging with the TRACE method, avoiding infinite loops. For example:

```
Max-Forwards: 5
```

This field can be ignored with all methods defined in HTTP configuration.

Proxy-Authorization field

This field allows the Client to authenticate itself (or its users) to an authorized station that requires delegation. This is the general syntax:

```
Proxy-Authorization: credentials
```

Proxy-Authorization field values ??include credentials that contain user agent credentials for the delegating station and / or the scope of the requested source.

Range field

The **Range** field identifies the internal range of the requested content from the document. The general syntax is:

```
Range: bytes-unit = first-byte-pos "-" [last-byte-pos]
```

First-byte-pos value in a byte-range-spec level byte-offset of the first byte in a sequence. The last-byte-pos value provides the byte-offset of the last byte in the sequence; that is, the byte positions are defined. You can specify a byte unit as bytes. The offset byte starts at 0. Some simple examples are as follows:

```
The first 500 bytes
Range: bytes = 0-499- The second 500 bytes
Range: bytes = 500-999- The 500 bytes final
Range: byte = -500- Ch? ??u tiên và byte m?i
Range: bytes = 0-0, -1
```

Many sequences can be listed, separated by commas. If the first digit in the byte sequence is distinguished by a forgotten comma, the sequence is assumed to be calculated from the end of the document. If the second digit is omitted, the sequence is the nth byte to the end of the document.

Referer field

This field allows the Client to determine the URI address of the source from which the URL was requested. The general syntax is as follows:

```
Referer: absoluteURI |  relativeURI
```

Here is an example:

```
Referer: http:///www.tutorialspoint.org./http/index.jsp
```

If the field value is a relational URI, it should be interpreted in relation to the Request-URI.

TE school

This field indicates any extension that transfer-coding is willing to accept in response and whether or not it is willing to accept the trailer field in a transfer-coding block. The following is the general syntax:

```
TE: t-codings
```

The presence of the keyword "trailers" indicates that the client is willing to accept trailer fields in a transfer-coding block and it is determined in one of two ways:

```
TE: deflate
TE:
TE: trailers, deflate; q = 0.5
```

If the TE field value is empty or no TE field is present, then only transfer-coding is closed (chunked). A notification with no transfer-coding is always acceptable.

User-Agent field

The **User-Agent field** contains information about the user agent that created the request. The following is the general syntax:

```
User-Agent: product |  comment
```

For example:

```
User-Agent: Mozilla / 4.0 (compatible; MSIE5.01; Windows NT)
```

Feedback fields from Server

Accept-Ranges

This field allows the Server to indicate its acceptance of the requested ranges for a source. The general syntax is:

```
Accept-Ranges: range-unit |  none
```

For example, a Server that accepts byte sequence requests can send:

```
Accept-Ranges: bytes
```

Server that does not accept any kind of request sequence for a source can send:

```
Accept-Ranges: none
```

This will advise the Client not to attempt to capture a range of requests.

School Age

Age School transfers the calculation of the amount of time since the response is generated at the sender's original Server. The general syntax is:

```
Age: delta-seconds
```

Age values ??are non-negative decimal integers, representing time in seconds. Here is a simple example:

```
Age: 1030
```

An HTTP / 1.1 Server that includes a hidden memory must include an Age field in each response created from its own hidden memory.

ETag school

The **Etag** field provides the current value of the object tag for the requested variation. The general syntax is:

```
Entity-tag ETag
```

Here are some simple examples:

```
ETag: "xyzzy"
ETag: W / "xyzzy"
ETag: ""
```

Field Location

The **Location** field is used to redirect the recipient to a location other than the Reqest-URI. The general syntax is:

```
Location: absoluteURI
```

Here is a simple example:

```
Location: http:///www.tutorialspoint.org./http/index.jsp
```

Other Content-Location field Location in which Content-Location identifies the original location of the object included in the request.

Proxy-Authenticate field

This field must be included as part of response 407. The general syntax is:

```
Proxy-Authenticate: challenge
```

Retry-After school

This field can be used with a 503 (Service Unavailable - service not available) response to indicate how long the service is expected to be unavailable to the client being requested. The general syntax is:

```
Retry-After: HTTP-date |  delta-seconds
```

Examples:

```
Retry-After: Fri, 31 Dec 1999 23:59:59 GMT
Retry-After: 120
```

In the following example, the delay is 2 minutes.

Server field

This field contains information about the software used by the original Server to control the request. The general syntax is:

```
Server: product |  comment
```

Here is an example:

```
Server: Apache / 2.2.14 (Win32)
```

If the response is being forwarded via an authorization station, the proxy application cannot modify the Server field.

Set-Cookie field

This field contains a name / value pair of information to retain for the URL. The general syntax is:

```
Set-Cookie: NAME = VALUE;  OPTIONS
```

Set-Cookie Feedback fields include Set-Cookie signs, followed by a comma-separated list of one or more cookies. Here are possible values ??that you can define as features:

Features and Description1 **Comment = comment**

This feature can be used to identify any comments associated with cookies.

2 **Domain = domain**

The Domain property specifies the domain with which the cookie is valid.

3 **Expires = Date-time**

The date the cookie will expire. If it is empty, the cookie expires when the user leaves the browser.

4 **Path = path**

The path attribute specifies the subset of the URLs this cookie applies.

5 **Secure**

It instructs the user agent to return cookies only in the form of a secure connection.

The following is an example of a simple cookie created by Server:

```
Set-Cookie: name1 = value1, name2 = value2;  Expires = Wed, 09 Jun 2021 10:18:1
```

Vary School

The **Vary** field specifies that the object has multiple sources and therefore can be in many ways to reach a specified list. The following is the general syntax:

```
Vary: field-name
```

You can specify multiple Headers separated by commas and a value that is a * without defining parameters (not limited to the required Headers). Here is a simple example:

```
Vary: Accept-Language, Accept-Encoding
```

Here, the school names are not sensitive.

WWW-Authenticate School

This field must be included in 401 response messages (not authorized). The field value consists of at least one challenge that instructs the proxy schema and the parameters that can be applied to the requested URI. The general syntax is:

```
WWW-Authenticate: challenge
```

Wall values ??can contain more than one challenge, or if more than one WWW-Authenticate field is provided, the contents of that challenge itself may contain a list separated by commas of authorization parameters.

Here is a simple example:

```
WWW-Authenticate: BASIC realm = "Admin"
```

Entity Headers

Allow school

This field lists the set of methods that are supported by the source confirmed by Request-URI. The general syntax is:

```
Allow: Method
```

You can define many methods that are distinguished by commas. Here is a simple example:

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

This field does not prevent a Client from trying other methods.

Content-Encoding field

This field is used as an editor to media type. The general syntax is:

```
Content-Encoding: content-coding
```

Content encryption is an attribute of an object defined by Request-URI. Here is a simple example:

```
Content-Encoding: gzip
```

If encrypting the content of an object is a request message that is not accepted by the source server, the server responds with a 415 status code (unsupported media type).

Content-Language field

This field describes the natural language of the reader intended for the included object. The following is the general syntax:

```
Content-Language: language-tag
```

Many languages ??can be listed for content that is intended for many readers. Here is a simple example:

```
Content-Language: mi
```

The first purpose of Content-Language is to allow a user to validate and differentiate objects according to the preferred language of the user.

Content-Length field

This field indicates the size of the object body, in the decimal number of system 8, is sent to the recipient or, in the case of the HEAD method, the size of the object body that will be sent, is required. is a GET. The general syntax is:

```
Content-Length: DIGITS
```

Here is a simple example:

```
Content-Length: 3495
```

Any Conten-Length greater than or equal to is an effective value.

Content-Location field

This field can be used to provide the source location for the object included in the message when the object is accessible from a separate location from a URI of the requested source. The general syntax is:

```
Content-Location: absoluteURI |  relativeURI
```

Here is a simple example:

```
Content-Location: http:///www.tutorialspoint.org./http/index.jsp
```

The Content-Location value also defines the base URI for the object.

Content-MD5 field

This field can be used to provide an object MD5 classification system for checking the integrity of information to recipients. The general syntax is:

```
Content-MD5: md5-digest using base64 of 128 bits MD5 digest as per RFC 1864
```

Here is a simple example:

```
Content-MD5: 8c2d46911f3f5a326455f0ed7a8ed3b3
```

The MD5 classification system is calculated based on the content of the entity body, including any content encryption that has been applied, but does not include any transmission encryption applied to Notification body.

Content-Range field

This field is sent with an internal entity body to determine where in the entire body part the internal body should be applied. The general syntax is:

```
Content-Range: bytes-unit SP first-byte-pos "-" last-byte-pos
```

Examples of byte-content-range-spec values, assuming that the object contains a sum of 1234 bytes:

```
The first 500 bytes:
Content-Range: bytes 0-499 / 1234- The second 500 bytes:
Content-Range: bytes 500-999 / 1234- All except for the first 500 bytes:
Content-Range: bytes 500-1233 / 1234- The last 500 bytes:
Content-Range: bytes 734-1233 / 1234
```

When an HTTP message consists of the content of a single sequence, this content is transmitted with a Content-Range, and a Content-Length indicates the actual number of bytes transmitted. For example:

```
HTTP / 1.1 206 Partial content
Date: Wed, 15 Nov 1995 06:25:24 GMT
Last-Modified: Wed, 15 Nov 1995 04:58:08 GMT
Content-Range: bytes 21010-47021 / 47022
Content-Length: 26012
Content-Type: image / gif
```

Content-Type field

This field instructs the media type of the object body to be sent to the recipient or, in the case of the HEAD method, the type of media that will be sent, a request is GET. The general syntax is:

```
Content-Type: media-type
```

Here is an example:

```
Content-Type: text / html;  charset = ISO-8859-4
```

Expires School

This field provides a Date / Time after which the response is said to expire. General syntax:

```
Expires: HTTP-date
```

Here is an example:

```
Expires: Thu, 01 Dec 1994 16:00:00 GMT
```

Last-Modified School

This field indicates the date and time at which the Server originally believed the transformation was last modified. The general syntax is:

```
Last-Modified: HTTP-date
```

Here is an example:

```
Last-Modified: Tue, 15 Nov 1994 12:45:26 GMT
```

According to Tutorialspoint

Previous post: Encrypt status in HTTP

Next lesson: Caching in HTTP