

# Lista degli header HTTP (rfc2616,rfc2068,rfc2617,rfc2069)

## REQUEST

### Host:

Specifica il nome di dominio del server su cui è presente la risorsa richiesta e la porta TCP su cui questo è in ascolto. Se non è specificata alcuna porta, si assume la porta standard del servizio a cui si fa riferimento (http porta 80, ftp porta 21 etc..). Il suo valore è estrapolato dall'URL (uniform resource locator) fornito dall'utente o contenuto nella referring resource (risorsa "di provenienza" la quale contiene un collegamento che permette di accedere ad un'altra risorsa - solitamente è un URL http). Nonostante i nomi di dominio siano "case-insensitive", talvolta il suo valore è interpretato in maniera "case-sensitive". E' un header indispensabile per l'implementazione del "virtual hosting", ovvero un metodo che permette di associare domain name multipli ad un unico server (o ad un pool di server). A partire dalla versione HTTP 1.1 va inserito obbligatoriamente in ogni request. Se assente, il server http risponde con uno status code 400 (Bad request).

ESEMPIO:

Host: www.w3.org

Host: www.w3.org:80

### Connection:

Specifica che tipo di connessione l'user agent (il client) vuole instaurare con il server (persistente o non persistente). In particolare, esplicita le opzioni che sono desiderate per la sessione attuale e, pertanto, il suo valore NON deve essere memorizzato da un eventuale proxy cache e riproposto nelle connessioni successive (Hop-by-hop header). Se la comunicazione avviene tramite protocollo HTTP 1.1, la connessione è implicitamente persistente e, pertanto, l'unica opzione utilizzata risulta "Connection: close" la quale indica che la connessione non è più persistente dopo la request corrente. Con il protocollo HTTP 1.0, invece, è necessario esplicitare se la connessione attuale è persistente o meno attraverso l'opzione "Connection: Keep-Alive". Con il protocollo 1.0, però, NON è possibile utilizzare il chunked transfer-encoding e deve, quindi, essere sempre specificato il Content-Length. Quando viene trasmesso un header "Connection: Keep-Alive" PUO' essere incluso l'header "Keep-Alive" (non è obbligatorio) in cui sono specificate delle opzioni aggiuntive. RICORDA: 1.1 → è implicito il Keep-Alive / 1.0 → è implicito il close

ESEMPLI:

Connection: close

Connection: Keep-Alive

### Accept:

Fa parte della famiglia degli "Header Accept". Specifica al server i "Content-Types" che sono accettati come risposta. Solo le risorse con tipo di dato presente in questo header possono essere inviate dal server. I valori assunti seguono la specifica dei **tipi MIME** (coppia oggetto/formato o type/subtype). Se il server non può inviare un response che soddisfi l'Accept, invia uno status code 406 (not acceptable). I valori più comuni sono i seguenti:

\*/\* → Sono accettati tutti i tipi MIME. Se "Accept" non è presente, per default si assume tale valore

oggetto/\* → Sono accettati tutti i formati di un determinato tipo di oggetto

text/html → pagina html || text/plain → file di testo generico non formattato || text/x-dvi → documento LaTeX

text/x-c → file testuale sorgente c || image/jpeg → immagine jpg || image/gif → immagine gif

audio/basic → µ-law audio || audio/mpeg → mp3 o altri formati mpeg || application/xml → risorsa XML

Oltre alla specifica del tipo MIME, è possibile specificare il parametro aggiuntivo "q" chiamato "quality factor" o "quality value". Esso può assumere un valore tra 0 e 1 e permette di specificare una gerarchia di priorità (quale risorsa inviare prima e quale dopo o quale risorsa inviare in sostituzione di un'altra). Quando "q" non è specificato, vale 1 per default.

**ATTENZIONE! Il significato del parametro "q", ovvero "valore di qualità", può essere fuorviante! Difatti tale parametro non ha nulla a che vedere con l'effettiva qualità della risorsa che si sta considerando!**

**La semantica ricollegata al nome del parametro può al più essere presa in considerazione come criterio logico per l'assegnazione di un valore di quality value ad un determinato tipo MIME.**

**Si consideri ad esempio un server che svolge la funzionalità di streaming audio. Su tale server è installato un software di encoding/decoding MP3. Tale software ha ottime prestazioni a livello computazionale sulla macchina.**

**Oltre a questo, è presente un software di elaborazione di file video AVI che, tra le sue funzionalità, permette di estrapolare la traccia audio di un video. Ovviamente le prestazioni di quest'ultimo software sono, in generale, peggiori di quelle del semplice software MP3.**

**Se un client si trova, quindi, a richiedere un file audio MP3 di una canzone a cui è anche ricollegato un video musicale AVI, nel momento in cui il software MP3 si arresta per un malfunzionamento o il file MP3 è corrotto, si può pensare di utilizzare la funzionalità di estrarre lo stesso audio dal file AVI.**

**Lo scenario sopra descritto permette, quindi, di concludere che il quality value maggiore è sicuramente da assegnare al tipo di file MP3 e quello minore alla traccia audio del file AVI poichè tra i due il server è stato principalmente pensato per il primo tipo ed ha prestazione maggiori con questo.**

Consideriamo il seguente esempio:

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

Il client accetta risorse di tipo text/html e text/x-c. Se queste non sono disponibili, allora accetta la tipologia text/x-dvi. Se, però, anche questa non è disponibile allora accetta text/plain.

Altro esempio:

```
Accept: text/*, text/html, */*
```

In questa situazione, vige la regola per cui i tipi MIME di medesimo oggetto si “sovrascrivono” e quello più specifico ha la precedenza. In questo caso, quello che ha maggiore precedenza è text/html poi, in sua mancanza, text/\* ed infine \*/\*.

Ultimo esempio:

```
Accept: text/*;q=0.3, text/html, image/jpeg;q=0.5, */*;q=0.4
```

Per questo esempio applichiamo il principio per cui il valore di “q” associato ad un tipo MIME, il quale non è specificato esplicitamente nel header accept, si determina ricercando la coppia oggetto/formato specificata nell'header che ha maggior precedenza (riferendoci alla logica della “sovrascrizione”) fra tutti i tipi con stesso oggetto del tipo MIME di cui si vuole conoscere il quality factor.

Assumiamo, ad esempio, che sul server siano presenti due risorse rispettivamente di tipo “image/gif” ed “text/plain” ed esaminiamo entrambe le situazioni. I valori di “q” associati ad ogni tipo risultano i seguenti:

text/html	q= 1
image/jpeg	q= 0.5
image/gif	q= 0.4
text/plain	q=0.3

Come è possibile notare, “text/plain” eredita il valore del suo “q” dal tipo MIME “text/\*” poichè di tutti i tipi MIME esplicitati che sono coerenti con “text/plain” (ovvero “\*/\*” e “text/\*”) risulta essere il più specifico.

Pertanto il client accetta la risorsa di tipo text/html. In sua mancanza, procede ad accettare anche image/jpeg. A questo punto, seguendo la logica del quality factor più basso, il client accetta “image/gif” e, in mancanza di questo, text/plain.

## Accept-Language:

Specifica i linguaggi naturali umani accettati dal client. Fa parte della famiglia “Header Accept” e, pertanto, svolge una funzionalità simile all'header “Accept”, focalizzandosi però solo sul linguaggio. E' possibile specificare il parametro “relative quality value” anche per questo header. Il carattere “\*” indica che sono accettati tutti i linguaggi. Se l'header non è presente, il server assume che tutti i linguaggi sono accettati dal client. I valori che l'header può assumere sono i “language-tag” definiti dallo IETF (Internet Engineering Task Force - rfc5646 ). Nel caso in cui sia presente un valore che non corrisponde a nessuno di questi tag, il suo quality factor è assunto pari a zero (viene ignorato dal server).

ESEMPLI:

Accept-Language: en-GB, english

In questo esempio, il client accetta il linguaggio inglese britannico. Il secondo valore non coincide con nessun tag definito dallo IETF ed è ignorato.

Accept-Language: it-IT;q=0.8,en-US;q=0.6,en;q=0.4

Il linguaggio accettato preferito è l'italiano italia ma, in mancanza di questo, sono accettati anche tutti gli altri tipi di italiano. Se l'italiano non è disponibile in toto, viene accettato l'inglese americano (US) e, infine, un qualsiasi tipo di inglese se anche quest'ultimo è assente.

## Accept-Encoding:

E' un “header accept”. Specifica i tipi di codifiche di compressione accettate dal client. Tale codifica è utilizzata per ridurre le dimensioni dei file interscambiati tra client e server. I tipi di compressione più comuni sono i seguenti:

gzip o x-gzip → tipo di codifica utilizzata dal programma di compressione “gzip”

compress o x-compress → formato di compressione prodotto dal programma UNIX “compress”

deflate → formato di compressione zlib, basato a sua volta sull'algoritmo “deflate”

identity → indica che non deve essere utilizzata nessuna compressione

sdch → Shared Dictionary Compression Over HTTP. E' un algoritmo di compressione creato ed utilizzato da Google.

Se tale header non è inviato, il server assume che sono accettati tutti i tipi di compressione. Se il server non può inviare le risorse nel formato di compressione richiesto, invia un errore 406 (Not acceptable). Non prevede l'utilizzo del quality value.

ESEMPIO:

Accept-Encoding: gzip,deflate,sdch

## User-Agent:

Specifica la stringa identificativa dello user agent. Tale stringa contiene informazioni sullo user agent che ha generato la request. E' utilizzato dal server per riconoscere il client e prendere le giuste contromisure legate agli eventuali limiti dello user agent (si veda IE6 che necessita di misure specifiche o i mobile browser per i quali è necessario sviluppare siti web ad hoc). La grande maggioranza degli user agent utilizza una stringa che include al suo interno la specifica del browser “Mozilla 5.0”, questo perché in passato molti server erano configurati per ignorare le richieste provenienti da altri tipi di browser. In generale, la stringa è composta dalle seguenti parti:

*Mozilla/[version] ([system and browser information]) [platform] ([platform details]) [extensions]*

ESEMPLI:

Google Chrome 31.0.16 su Windows 7 → Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/31.0.1650.63 Safari/537.36

Internet Explorer 9 su Windows 7 → Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Trident/5.0; BOIE9;ENUSMSNIP)

Firefox 3.6 su Windows 7 → Mozilla/5.0 (Windows; U; Windows NT 6.1; it; rv:1.9.2.28) Gecko/20120306 Firefox/3.6.28 (.NET CLR 3.5.30729; .NET4.0E)

Internet Explorer 8 su Windows XP → User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; .NET4.0C; .NET4.0E)

Firefox 3.0.10 su Ubuntu → Mozilla/5.0 (X11; U; Linux i686; it; rv:1.9.0.10) Gecko/2009042513 Ubuntu/8.04 (hardy) Firefox/3.0.10

Opera 18 su Windows XP → User-Agent: Mozilla/5.0 (Windows NT 5.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/31.0.1650.63 Safari/537.36 OPR/18.0.1284.68

Safari 5 su Windows XP → User-Agent: Mozilla/5.0 (Windows NT 5.1) AppleWebKit/534.57.2 (KHTML, like Gecko) Version/5.1.7 Safari/534.57.2

## Authorization:

E' utilizzato per implementare i meccanismi di autenticazione HTTP. Tali meccanismi sono necessari per il controllo degli accessi a determinate risorse, in cui è obbligatorio che lo user agent si “faccia riconoscere” dal server. Solitamente è utilizzato quando uno user agent vuole accedere ad un'area riservata, chiamata “**realm**” o “dominio di autenticazione” o “dominio di protezione”, e alle risorse in essa contenute. Le procedure di autenticazione che prenderemo in considerazione sono del tipo “challenge-response” (letteralmente “sfida-risposta”).

Inizialmente lo user agent invia una normale richiesta di una risorsa al server senza esplicitare l'header authorization. Se tale risorsa appartiene ad un realm protetto, il server invia un “challenge” (una response http) con uno status code 401 (Unauthorized) ed un particolare header chiamato “WWW-Authenticate”. A questo punto, lo user agent richiede all'utente le credenziali di accesso e le invia in una “response” (una request http) includendo l'header authorization.

*Il challenge sostanzialmente rappresenta un insieme di informazioni che vengono inviate dal server ad uno user agent che invia una richiesta di accesso ad un realm (area riservata) senza che risulti autorizzato all'accesso. Esso è nella pratica costituito dall'header WWW-Authenticate e da tutti i suoi parametri.*

Una volta che lo user agent risulta autenticato su un determinato realm, esso potrà accedere a tutte le risorse appartenenti a tale area riservata senza autenticarsi nuovamente.

Nell'header “Authentication” si specificano le informazioni di autenticazione del client, ovvero il tipo di tecnica di autenticazione in uso, le credenziali di accesso e altri parametri aggiuntivi. Le tecniche di autenticazione challenge-response più diffuse sono le seguenti:

*Basic Access Authentication:* E' stata introdotta con l'HTTP 1.0 e continua tutt'ora ad essere utilizzata con HTTP 1.1. E' un tipo di autenticazione in cui il valore del parametro “realm”, presente SOLO nel challenge, è una stringa usata dal server per differenziare le aree riservate nei vari challenge. Il server accetta la richiesta di autenticazione solo se le credenziali sono quelle corrette per l'area riservata identificata dal Request-URL (l'url della risorsa inserito nella request line). Una volta ricevuta una richiesta di accesso ad un realm da uno user agent non autorizzato, il server risponde con l'header:

```
WWW-Authenticate: Basic realm="<realmvalue>"
```

dove <realmvalue> è la stringa generata dal server per identificare lo spazio di protezione dell'URL richiesto. Una volta ricevuto il challenge, il client invia una request in cui sono presenti le credenziali chieste all'utente. Tali credenziali sono inviate come una stringa codificata in Base64 del formato “<userid>:<password>” con <user> e <password> richiesti all'utente. Questa informazione è, quindi, inserita nell'header “Authorization” che viene aggiunto dall'user agent all'invio della request. Il response, con ad esempio <userid>=Aladdin e <password>=“open sesame”, risulta:

```
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

La Basic Access Authentication è un metodo di autenticazione molto semplice, che non richiede l'utilizzo di cookies e sessioni ed usa solo header HTTP standard. E' da ritenersi una tecnica con basso livello di sicurezza, il cui scopo principale è il mero interscambio di credenziali tra client e server. Il trasferimento delle informazioni, infatti, è sostanzialmente in chiaro, cioè senza alcun algoritmo di crittografia efficace che possa garantire la **confidenzialità** delle informazioni (la confidenzialità consiste nella protezione dei dati scambiati tra due interlocutori in modo tale che una parte terza non possa carpire i dati che essi si inviano). Per questa ragione, se è necessario garantire anche un buon livello di sicurezza dei dati, è doveroso utilizzare la tecnica su protocollo HTTPS.

*Digest Access Authentication:* E' stato introdotto l'HTTP 1.1 ma è ad ogni modo implementato anche da server e user agent con protocollo 1.0 poiché già questo fornisce tutte le direttive di base su cui poter implementare efficacemente questo tipo di metodo. E' bene specificare che questo metodo non è da considerarsi una soluzione completa alle

esigenze di sicurezza sul web, in quanto non implica l'invio protetto del contenuto dei messaggi scambiati tra client e server, ma solo di quelle informazioni relative al protocollo di autenticazione ed, in particolare, le password che, al contrario del Basic Access in cui vengono inviate in chiaro, sono crittografate. **Il problema principale della Digest, quindi è questo: le password passano codificate e, quindi, non possono essere carpite in modo fraudolento, ma tutte le informazioni sulla transazione sono in chiaro e possono essere “rubate”. Questo, quindi, permette di poter poi attuare attacchi di tipo “replay”, cioè di terzi che si intromettono nella comunicazione.**

**A questa problematica si accompagna un'ulteriore constatazione, ovvero che questo protocollo non prende in considerazione nessuna disposizione di sicurezza iniziale tra client e server per definire la password utente. Quest'ultima, infatti, deve essere fatta arrivare la prima volta sul server, in modo tale che questo possa memorizzarla sui suoi spazi di storage e creare una utenza! E' quella che potremmo chiamare “fase di registrazione”.**

La Digest Access Authentication prevede l'utilizzo del metodo di crittografia MD5. Esso è costituito da una funzione di hashing la quale, applicata ad un certo insieme di dati, restituisce un valore di lunghezza fissa a 16 Byte espresso come stringa di 32 caratteri a valori esadecimali. Questo tipo di metodo è di tipo “one way”, ovvero è facile da codificare ma difficile da decodificare se non si conoscono gli input su cui è stata fatta la codifica. Le informazioni, una volta crittografate, sono chiamate “**fingerprint**” (letteralmente “impronta digitale”).

La challenge (il valore dell'header WWW-Authenticate) da parte del server prevede l'utilizzo di alcuni parametri specifici. I principali sono illustrati di seguito:

**realm** → E' una stringa che il client mostra all'utente, in modo tale che questo sappia che username e password deve inserire. Deve contenere almeno il nome dell'host su cui è presente l'area riservata alla quale si vuole accedere.

**nonce** (pronunciato “nouans”) → E' una stringa generata dal server. E' utilizzata per prevenire gli attacchi di tipo “replay-attack” che, al contrario dei tipi di attacco “man in the middle attack”, possono avvenire anche in modo asincrono, cioè quando la comunicazione è terminata. Essa consiste nella codifica Base64 della stringa seguente:

<timestamp>:<Etag>:<private-key>

dove <timestamp> rappresenta l'ora attuale generata dal server, <Etag> è il valore dell'header “Etag” e <private-key> un valore generato dal server. Il client restituisce la stringa nonce al server così come l'ha ricevuta.

L'utilizzo del valore <timestamp> permette di limitare gli attacchi di tipo “replay”. Difatti una volta che il client ha restituito il nonce, il server ricalcola un nuovo time-stamp sull'ora corrente, decodifica la stringa e confronta il valore appena calcolato con quello decodificato. Se è passato troppo tempo, può decidere di non validare la richiesta di autenticazione. Il valore di <Etag>, invece, permette di scartare immediatamente una richiesta di autenticazione su una risorsa il cui “Etag” è cambiato, cioè che risulta avere una versione aggiornata sul server. Il **nonce** viene ricalcolato ad ogni nuova richiesta di accesso ad un'area riservata.

**opaque** (pronuncia “opeik”) → E' una stringa generata dal server il cui formato segue la codifica Base64. Essa deve essere restituita dal client così come gli viene consegnata. Continua ad essere inviata nell'header “Authorization” per tutti le richieste successive all'autenticazione che fanno riferimento ad URL di uno stesso dominio di validità.

Oltre a questi parametri base, è possibile aggiungerne altri al fine di migliorare ulteriormente la sicurezza (qop).

La response (l'header Authorization) da parte del client prevede l'utilizzo di alcuni parametri seguenti:

**username** → Stringa rappresentate lo username. E' chiesta all'utente.

**realm** → Stringa identica a quella inviata dal server del suo parametro “realm”

**nonce** → Stringa nonce identica a quella inviata dal server

**uri** → L'URL della risorsa a cui si vuole accedere, come specificato del “Request-URL” della “Request-Line”. Tale valore viene duplicato poiché un eventuale server proxy potrebbe modificare la “Request-Line”.

**response** → Rappresenta la “fingerprint” di cui si è prima parlato. Per evitare l'abuso di circolazione delle password, oltre alle password vengono codificate anche altre informazioni. Il response è così ottenuto:

response = MD5( MD5(< username>:<realm>:<password> ):<nonce>:MD5 (<method>:<URL> ) )

dove <username>, <realm> e <password> sono rispettivamente i valori dello username, password e realm, <nonce> il valore del nonce inviato dal server, <method> il metodo HTTP fatto dalla request (GET, POST, HEAD) e <URL> l'URL della risorsa come specificato del parametro “uri”. Tale formato può variare a seconda dei parametri aggiuntivi che il server può inviare oltre a quelli elencati sopra(cnonce,nc).

Il server, una volta ricevuto il response dal client, calcola a sua volta la stessa stringa MD5 con i valori che lui ha a disposizione, tra cui la username e password corrette per il realm al quale il client vuole accedere. Vien da sé che se l'utente ha inserito le credenziali corrette, tale calcolo fatto dal server coincide con il response inviato dal client. In termini reali, viene effettuata una divisione polinomiale tra "response" inviato dal client e stringa codificata MD5 calcolata dal server: se la divisione ha resto zero, i polinomi sono uguali e quindi le credenziali sono corrette.

**opaque** → Stringa opaque identica a quella inviata dal server in precedenza

#### ESEMPIO

Una volta ricevuta una richiesta di accesso ad un realm da uno user agent non autorizzato, il server risponde con l'header seguente:

```
WWW-Authenticate: Digest realm="testrealm@host.com",
                    nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
                    opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

Il client, ricevuto tale header, risponde con:

```
Authorization: Digest username="Mufasa",
                      realm="testrealm@host.com",
                      nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
                      uri="/dir/index.html",
                      response="6629fae49393a05397450978507c4ef1",
                      opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

**IMPORTANTE:** Il parametro/direttiva "realm" (di tipo case-insensitive quindi si può anche scrivere "REALM") è richiesta per tutti gli schemi di autenticazione di tipo challenge-response. Bisogna, però, sottolineare una differenza: nella BA, "realm" non ha alcuna funzione "attiva" per la fase di validazione ed è semplicemente una stringa che viene mostrata all'utente per fargli capire che credenziali digitare; nella DA, invece, il "realm" fa parte dei parametri che poi vengono codificati all'interno del "response" e che, quindi, contribuiscono a validare tutta la sessione di autorizzazione.

#### Date:

Specifica la data e l'ora di invio del messaggio HTTP. Come header facente parte di una request, può essere inserito quando la richiesta include un entity-body, come ad esempio con il metodo POST, ma non è ad ogni modo obbligatorio. Non deve mai essere inviato se il client non ha un orologio interno. Il formato della stringa rappresentante il valore di "Date" DEVE essere un HTTP-date, definito nell'RFC 1123. La sintassi è la seguente:

<giornosettimana>, <giornomese> <mese> <anno> <ore>:<minuti>:<secondi> GMT

E' necessario che la stringa corrisponda perfettamente a quanto illustrato, considerando anche gli spazi e tutti i segni di punteggiatura. Di seguito sono esplicitati tutti i valori presenti.

<giornosettimana> è costituito da un'abbreviazione del giorno della settimana in lingua inglese. Il valore deve essere uno dei seguenti: "Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"

<giornomese> è il numero rappresentante il giorno del mese. Se inferiore a dieci, va aggiunto uno zero prima del numero

<mese> è un'abbreviazione del mese dell'anno in lingua inglese. Deve essere uno dei seguenti: "Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"

<anno> è il valore dell'anno con 4 cifre.

<ore>, <minuti>, <secondi> sono valori numerici indicanti l'ora, i minuti ed i secondi. Sono indicati sempre con due cifre, quindi se uno dei valori è inferiore a dieci va inserito uno zero.

#### ESEMPLI:

Date: Sun, 06 Nov 1994 08:49:37 GMT

Date: Tue, 15 Nov 1994 08:12:31 GMT

Date: Sun, 12 Jan 2014 11:11:59 GMT

## Range:

Permette di recuperare solo una parte della risorsa richiesta. E' un header che può essere inserito quando il metodo della request è un GET. Viene specificato il range di valori che si vuole effettivamente ricevere dal server. Per quanto l'utilizzo dell'intestazione non sia obbligatorio, è bene che un server implementi tale header poiché permette un recupero efficiente di informazioni quando una parte di un messaggio inviato è andata perduta e non è, pertanto, necessario richiederne il re-invio nella sua interezza. Se il server supporta tale header e il range specificato è coerente con la risorsa richiesta, il server restituisce la risorsa parziale con status code 206 (Partial Content). Se sono presenti anche header condizionali, il server restituisce status code 206 solo se la condizione è soddisfatta. Se l'intervallo non è coerente con la risorsa, il server deve restituire lo status code 416 (Requested range not satisfiable).

Il tipo di range accettato è il "byte range", ovvero un intervallo di byte. Il primo byte parte da zero, ma è possibile specificare un offset per l'inizio dell'intervallo da considerare. E' possibile specificare un solo intervallo di byte o più intervalli. Gli intervalli specificati sono inclusivi, ovvero anche i byte che delimitano l'intervallo ne fanno parte.

### ESEMPI

Assumendo un entity-body di lunghezza 10000, l'intervallo di byte che lo rappresenta interamente è 0-9999.

Range: bytes=0-499 → primi 500 byte

Range: bytes=500-999 → secondi 500 byte

Range: bytes=-500 → i 500 byte finali

Range: bytes=9500- → i 500 byte finali (forma alternativa)

Range: bytes=0-0,-1 → il primo l'ultimo byte

## If-Range:

Viene utilizzato in aggiunta all'header "Range". Se inviato da solo deve essere ignorato dal server. Il suo valore è rappresentato dall'header "Etag" assegnato dal server alla risorsa richiesta o, in mancanza di questo, può usare la data specificata nell'header "Last-Modified". E' un header condizionale il quale specifica al server di inviare il range di byte richiesto nell'intestazione "Range" SOLO SE la risorsa non è cambiata rispetto a quella posseduta dal client (ovvero Etag o Last-Modified non sono cambiati), altrimenti esso deve inviare INTERAMENTE la risorsa aggiornata. Se questa condizione non è soddisfatta, quando il server invia la risorsa nella sua interezza deve inviare uno status code 200 (OK).

### ESEMPIO

If-Range: "737060cd8c284d8af7ad3082f209582d"

## Referer:

E' un header che permette al client di specificare al server l'URL della risorsa da cui si è ottenuto il "Request-URL" della request corrente. In questo modo, il server può rendersi conto da dove è stata originata la richiesta e tale informazione può essere utile per alcune ottimizzazioni (per esempio ottimizzare il caching) o a scopi statistici e promozionali (con il logging dei referer). Un esempio comune si ha quando un utente clicca su un collegamento ipertestuale in un browser e quest'ultimo invia una request al server avente la pagina web di destinazione: in questo caso la request include l'intestazione "referer" che indica l'ultima pagina su cui è stato l'utente.

Nel caso in cui la "Request-URI" è ottenuta da una risorsa che non possiede un proprio URL (ad esempio quando l'utente digita manualmente l'URL della pagina alla quale vuole accedere) NON DEVE essere aggiunta tale intestazione. Se il valore specificato è un URL relativo (non viene indicato tutto il percorso della risorsa) lo si deve riferire al "Request-URL" della request line ( se il referer è "/personaggi/pippo.html" e il request-url "/disney/", l'URL della risorsa di referring sarà "/disney/personaggi/pippo.html").

### ESEMPIO:

Referer: <http://www.w3.org/hypertext/DataSources/Overview.html>

## **From:**

Permette di specificare un indirizzo e-mail nel formato “Addr-spec” (definito nell'RFC2822 par. 3.4.1). Tale indirizzo dovrebbe essere quello della persona che controlla l'user agent da cui è stata originata la richiesta. Un esempio concreto di utilizzo si ha quando la richiesta viene effettuata da un “web robot” (applicazione web che svolge operazioni automatizzate su internet) ed è quindi bene che questo specifichi l'header “Form” in modo tale che le persone che gestiscono il robot possano essere contattate in caso di problemi. Il client non dovrebbe inviare questa intestazione senza il consenso dell'utente.

### **ESEMPIO**

From: webmaster@w3.org

## **If-Modified-Since:**

E' una intestazione condizionale. In particolare, la condizione permette di specificare che se la risorsa richiesta E' STATA modificata successivamente alla data specificata come valore dell'header allora il server può applicare la request del client (status code 200 se è una GET). Il server ha a disposizione la data dell'ultima modifica effettuata su una risorsa: se questa data è successiva a quella inviata in questo header, allora il server deve inviare la risorsa.

In caso contrario, il server deve inviare uno status code 304 (not modified) senza alcun entity-body (il client ricarica quella risorsa dalla sua cache). Il formato della data è di tipo HTTP-date (vedi specifica nell'header “date”). Lo scopo principale di questa intestazione è di rendere efficiente l'utilizzo e l'aggiornamento della cache del client minimizzando il numero di transazioni. Se è presente anche un header “Range” la condizione viene comunque applicata ma l'insieme di dati su cui il server deve ragionare è solo quello specificato dall'intestazione “Range”.

### **ESEMPIO**

If-Modified-Since: Sat, 29 Oct 1994 19:43:31 GMT

## **If-Unmodified-Since:**

E' un header condizionale. E' la condizione opposta dell'header “If-Modified-Since”. In particolare, la presenza di questo header specifica al server che se la risorsa richiesta NON E' STATA modificata rispetto alla data specificata come valore dell'header, allora il server deve comportarsi come se questa condizione non fosse presente (quindi inviare la risorsa con status code 200 se è stata fatta una GET). In caso contrario, quindi nel caso in cui la risorsa risulta essere stata modificata dopo la data specificata, l'operazione richiesta dal client non deve essere portata a compimento ed il server deve rispondere con uno status code 412 (Precondition Failed).

### **ESEMPIO:**

If-Unmodified-Since: Sat, 29 Oct 1994 19:43:31 GMT

## **If-Match:**

E' una intestazione condizionale. E' utilizzato dal client per se una risorsa in suo possesso è aggiornata oppure no. I valori da specificare sono degli dell'entity tag oppure il carattere “\*”, indicante un valore qualsiasi. E' possibile specificare anche più di un entity-tag. La condizione prevede che se uno qualsiasi dei valori di entity tag coincide con l'entity tag della risorsa richiesta o se è stato specificato il carattere “\*” e l'entity tag per la risorsa esiste, allora il server deve eseguire la request come se questo tag non ci fosse. In caso contrario, cioè se non c'è riscontro tra gli entity tag o se dato il carattere “\*” l'Etag non esiste, il server deve restituire uno status code 412 (Precondition Failed).

Uno degli utilizzi maggiori di questo header viene fatto per evitare la modifica errata di una risorsa quando questa risulta più aggiornata di quella posseduta dal client. Si consideri, ad esempio, un client che invia una request di aggiornamento con metodo POST. Senza questo header la risorsa verrebbe aggiornata a priori, anche nel caso in cui il server fosse in possesso di una versione modificata non in possesso del client.

### **ESEMPIO:**

If-Match: "xyzyz"  
If-Match: "xyzyz", "r2d2xxxx", "c3piozzzz"  
If-Match: \*



## If-None-Match:

E' una intestazione condizionale con logica opposta all'"If-Match". Quindi, se NESSUNO dei valori di Etag specificati in questa intestazione coincide con l'Etag della risorsa richiesta o se è specificato il carattere "\*" e l'Etag della risorsa NON esiste, allora il server deve eseguire la request inviata come se questo tag non fosse presente ed ignorare anche eventuali "If-Modified-Since" presenti. Pertanto, il server NON DEVE rispondere con uno status code 304 (Not modified)

Contrariamente, se uno dei valori coincide con l'Etag della risorsa o se specificato il carattere "\*" l'Etag esiste, allora il server NON DEVE eseguire la request A MENO CHE non risulti soddisfatto un "If-Modified-Since" eventualmente presente. Quando la condizione non è soddisfatta, se la request è una GET o una HEAD, il server DOVREBBE rispondere con uno status code 304 (Not modified) con le intestazioni memorizzate in cache (in particolare l'entity tag della risorsa che coincide con uno dei valori specificati). Per tutti gli altri metodi, il server deve rispondere con lo status code 412 (Precondition Failed).

### ESEMPI

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

**NOTA: If-Match/If-None-Match e If-Modified-Since/If-Unmodified-Since possono essere utilizzati per avere gli stessi riscontri condizionali. Ciò che cambia è il valore che vanno a confrontare: i primi confrontano delle date e, quindi, una modifica "cronologica" delle risorse, i secondi confrontano gli Etag che possono essere pensati come lo stato di una risorsa in un determinato momento. Specificare più Etag in uno stesso header ha il significato di comunicare al server che possono essere diversi gli stati di una risorsa che il client accetta.**

## Pragma:

Questo header è utilizzato per esplicitare delle direttive che devono essere recepite e applicate lungo tutta la catena request/response. In altre parole, tale header deve passare attraverso i proxy e le applicazioni gateway che fanno da tramite tra client e server durante le loro sessioni di comunicazione e applicare le direttive esplicitate in Pragma. Non è possibile assegnare direttive diverse per ogni entità presente lungo la catena.

L'unica direttiva accettata per questa intestazione è "no-cache". Quando questa è presente in una request, il client si comporta inoltrando la richiesta al server di origine indipendentemente dal fatto che l' user agent abbia o meno una copia di quella risorsa salvata in cache. E' una direttiva che porta allo stesso comportamento dell'header "Cache-Control: no-cache", introdotto con HTTP 1.1. Pertanto, per una questione di retro-compatibilità con HTTP 1.0, è bene specificare assieme sia "Pragma" che "Cache-Control". Se un server o client http1.1 ricevono un messaggio con il solo "Pragma: no-cache" devono interpretarlo come se fosse in "Cache-Control: no-cache".

### ESEMPIO:

Pragma: no-cache

## Cache-Control:

E' usato per specificare delle direttive che DEVONO essere rispettate da tutti i meccanismi di caching presenti lungo la catena request/response. Vedi l'header nel messaggio di response.

## Cookie:

E' descritto nell'rfc2109. Permette di gestire i cookie, ovvero delle informazioni che client e server si scambiano nell'ambito di una sessione e che sono memorizzate dall'user agent in dei file. Essendo HTTP stateless (senza memoria dello stato in cui si trova), questi cookie permettono di memorizzare delle informazioni riguardanti la sessione attuale.

La creazione di una sessione con utilizzo di cookie è avviata per volontà del server, il quale invia una intestazione "Set-Cookie". Le informazioni giunte al client vengono memorizzate su un file gestito dall'user agent. Se il client decide di continuare quella sessione, invia a sua volta l'intestazione "Cookie" presente in questo file. Ogni volta che il client accede ad un determinato sito web, il cookie corrispondente viene da quest'ultimo inviato al server. A questo punto il server può decidere di ignorare l'header "Cookie" oppure di inviare nuovamente un "Set-Cookie" con informazioni uguali o differenti a quelle recapitategli dal client. Alla fine un server può decidere di terminare una sessione (e quindi invalidare il cookie del client) inviando un "Set-Cookie" con parametro "Max-Age=0". Il server può decidere di inviare

“Set-Cookie” multipli, ognuno dei quali è memorizzato in un diverso file dal client.

I parametri utilizzabili con l'header “Cookie” sono i seguenti:

**NAME=VALUE** → E' la stessa coppia NAME/VALUE specificata nel Set-Cookie del server.

**\$Version** → E' il valore dell'attributo “Version” presente nel Set-Cookie inviato dal server. Se non è presente, deve essere comunque esplicitato con valore zero.

**\$Path** → E' il valore di “Path” presente nel Set-Cookie del server. Se non presente, deve essere omissso.

**\$Domain** → E' il valore di “Domain” presente nel Set-Cookie del server. Se non presente, deve essere omissso.

In uno stesso header “Cookie” possono essere presenti più coppie NAME/VALUE ognuno con propri attributi.

I criteri con cui uno user agent sceglie un cookie piuttosto che un altro da inviare al server sono i seguenti:

- 1) Il valore dell'attributo “Domain” del cookie deve coincidere con l'host name del server al quale si sta inviando la request
- 2) Il valore dell'attributo “Path” del cookie deve coincidere con il prefisso del Request-URL
- 3) Il valore di Max-Age deve essere tale da non rendere il cookie scaduto. In questo caso, il client non lo invia.

E' da sottolineare che più cookie possono soddisfare questi criteri contemporaneamente.

ESEMPIO:

```
Cookie: $Version="1";  
       Customer="WILE_E_COYOTE"; $Path="/acme";  
       Part_Number="Rocket_Launcher_0001"; $Path="/acme";  
       Shipping="FedEx"; $Path="/acme"
```

**TE:**

E' un header di tipo hop by hop. Permette di specificare al server quali tipi di codifica sono ammessi dal client per il “Transfer-Encoding” e se accetta i “Trailers” (vedi Transfer-Encoding). E' possibile specificare anche un quality value.

ESEMPIO:

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

# RESPONSE

## Date:

Specifica la data e l'ora di invio del messaggio HTTP. Come header facente parte di una response, il server DEVE SEMPRE includerlo tra le intestazioni dei suoi messaggi di risposta a meno che il messaggio non rappresenti uno status code 500 (Internal Error) o 503 (Service unavailable). Per la sintassi, fare riferimento a quanto detto per lo stesso header nella request.

## Accept-Ranges:

Permette di specificare quali tipi di range sono accettati dal server (vedi l'intestazione "Range" della request). Non è un header obbligatorio anche nei casi in cui il server possa accettare un certo tipo di range. Difatti i client possono comunque inviare una richiesta parziale di una risorsa anche se il server non specifica questa intestazione. Se, però, il server non è in grado di accettare nessun tipo di range DOVREBBE specificare questa intestazione con il valore "none": in questo caso il client non dovrebbe inviare alcuna intestazione "Range". Nel caso in cui arrivi, il server deve rispondere con status code 416 (Requested range not satisfiable).

ESEMPI

Accept-Ranges: bytes

Accept-Ranges: none

## Connection:

Stessa funzionalità del request. Per i server Apache, l'header "Keep-Alive" prevede le opzioni "timeout" (indica il tempo in secondi oltre il quale si fa decadere la connessione se non giungono altri messaggi) e "max" (indica il numero massimo di messaggi che possono essere inviati al server nell'ambito della stessa connessione) con valori di default rispettivamente pari a 5 secondi e 100 connessioni massime.

ESEMPI:

Keep-Alive: timeout=5, max=100

Connection: Keep-Alive

## WWW-Authenticate:

E' utilizzato per implementare i meccanismi di autenticazione HTTP. Si specificano le informazioni per richiedere al client di autenticarsi, ovvero il tipo di tecnica di autenticazione in uso e altri parametri aggiuntivi. Vedi l'header "Authorization".

## Set-Cookie:

E' descritto nell'RFC2109. Permette di gestire le informazioni, presenti nei file cookie, che client e server si scambiano nell'ambito di una sessione. Viene aggiunto nelle intestazioni del server per settare i parametri di un cookie inerente una determinata sessione.

I parametri utilizzabili in "Set-Cookie" sono i seguenti:

**NAME=VALUE** → Parametro obbligatorio. Indica il nome dell'informazione da tenere memorizzata durante la sessione e il valore attribuito. Se, ad esempio, effettuo il login via web in un negozio online, il server mi potrebbe inviare una coppia del tipo *Cliente="MARIO\_ROSSI"*. Tale coppia NOME/VALORE è bene che abbia significato solo per il server, in quanto tutto il contenuto dei cookie viene trasmesso in chiaro e potrebbe essere carpito da un terzo.

**Comment** → Opzionale. E' una stringa che descrive in che modo il server intende utilizzare le informazioni scambiate tramite cookie. Il client può analizzare queste informazioni e decidere se continuare la sessione o meno.

**Domain** → Opzionale. Specifica il dominio per cui è valido il cookie.

**Path** → Opzionale. Indica l'URL o il subset di URL sul quale il cookie è da ritenersi valido.

**Max-Age** → Opzionale. Definisce il tempo di vita del cookie in secondi. Scaduto tale tempo, il cookie è da ritenersi

non valido e deve essere rigenerato.

**Secure** → Opzionale. Non ha un valore assegnato. Se presente, indica al client che è necessario proseguire la comunicazione utilizzando un metodo sicuro come, ad esempio, HTTPS (Secure Socket Layer)

**Version** → Obbligatorio. Indica la versione sotto cui il cookie è stato generato. Per la specifica presente nell'RFC2109 la versione è la 1.

ESEMPIO:

Set-Cookie: UserID=JohnDoe; Domain=docs.foo.com; Path=/accounts; Max-Age=3600; Version=1; Secure

## Expires:

Rientra tra le intestazioni per il controllo del caching. Permette di specificare la data oltre la quale il response di cui fa parte è da considerarsi scaduto. Il formato della data è HTTP-date (vedi header Date). Se è presente anche l'header "Cache-Control: max-age", questo sovrascrive l'Expires.

ESEMPIO:

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

## Pragma:

Vedi definizione nel response.

## Cache-Control:

Viene utilizzato per l'implementazione di direttive riguardanti il caching. Tali direttive DEVONO essere rispettate da tutti i meccanismi di caching presenti lungo la catena request/response. Tali direttive sono "unidirezionali", nel senso che il fatto che alcune direttive siano incluse in un messaggio non significa che debbano essere applicate anche al messaggio di risposta. E' implementato a partire da HTTP 1.1. Per l'HTTP1.0 la cache deve essere gestita solo con le intestazioni Expires, header condizionali (come If-Modified-Since) e Pragma.

Alcune delle direttive principali sono le seguenti:

**private** → Presente solo nelle response. Indica che la risposta messa in cache è utilizzabile solo per determinati utenti e non per tutti. Non deve, quindi, essere salvata in una cache condivisa (shared cache).

**s-maxage** → E' una direttiva presente solo nelle response. Se specificato, sovrascrive il valore dell'header "Expires" di "max-age". E' valido solo per la cache condivisa. Indica il tempo oltre il quale la risorsa è da ritenersi scaduta, in più applica una politica per cui i meccanismi di cache condivisa non devono MAI fornire in risposta un contenuto scaduto ma devono sempre rivalidarlo dal server di origine.

**max-age** → Valida sia per le request che per le response. E' un valore in secondi che indica che il client accetta il messaggio di risposta, inviato da un proxy di caching, solo se questo ha un'età in secondi inferiore a quella di questa direttiva. In caso contrario, la risorsa deve essere rivalidata. Il server di origine lo usa per comunicare al server di caching il tempo massimo di validità della risorsa (sovrascrivendo Expires).

**max-stale** → Valida solo per le request. Indica che il client accetta anche messaggi che hanno superato il loro tempo di scadenza. Se viene assegnato un valore, indica i secondi oltre la scadenza in cui il client è disposto ad accettare messaggi scaduti, al contrario il client accetta risposte scadute di qualsiasi età. Ogni volta che un server di caching invia una risorsa scaduta, deve aggiungere l'intestazione "Warning" con codice 110 (Response is stale).

**must-revalidate** → Valido solo per le response. E' una direttiva usata da un server per specificare al server di caching e a tutti gli altri meccanismi di cache che le risorse da loro memorizzate non devono essere mai spedite fino a quando non saranno rivalidate dal server di origine. Se il server di origine non risponde, il server di caching invierà un codice 504 (Gateway time-out) ad ogni richiesta effettuata per quella risorsa.

**no-cache** → Valido sia per le request che per le response. Quando specificato, obbliga tutti i meccanismi di cache intermedi a recuperare dall'interlocutore di origine la risorsa richiesta e, quindi, a rivalidarla.

**Proxy-revalidate** → Ha il medesimo significato di must-revalidate con la differenza che si applica solo a cache private.

I meccanismi sopra descritti fanno riferimento alla tipologia di cache-control "*Server-Specified expiration*" in cui è il server a comunicare a tutti i meccanismi di caching tramite direttiva **max-age** o tramite header "Expires." Se

nessuno di questi due è presente, entra in gioco la cosiddetta “*Heuristic Expiration*”. Tale tecnica indica che i tempi di scadenza sono calcolati in modo euristico, cioè con dei calcoli di stima ottimistica (la risorsa è valida secondo il caching server ma in realtà è scaduta) o pessimistica (la risorsa è scaduta per il cache server ma in realtà è ancora valida).

I meccanismi di caching, quando inviano risorse la cui scadenza è calcolata in modo euristico, devono aggiungere l'header “Warning” con codice 113 (Heuristic expiration).

ESEMPIO:

Cache-Control: no-cache

Cache-Control: max-age=3600

Cache-Control: max-age=0 → obbliga i meccanismi di cache intermedi a rivalidare la risorsa dal server di origine.

### **Warning:**

Permette di specificare un messaggio informativo aggiuntivo per comunicare possibili problemi inerenti l'entity body. Possono essere inviate intestazioni Warning multiple. I codici di errore sono specificati nell'RFC2616.

ESEMPLI:

Warning: 199 Miscellaneous warning

### **Content-Range:**

Viene aggiunto in un response contenente un entity-body parziale. Specifica in quale posizione della risorsa completa deve essere collocata la risorsa parziale inviata. DEVE essere specificato solo un intervallo di byte e DEVE specificare sempre la posizione assoluta del primo e dell'ultimo byte dell'intervallo (nel “Range” era possibile non specificare l'ultimo byte di un intervallo se questo coincideva con la lunghezza della risorsa. In questa header devo sempre specificare la posizione). Inoltre DOVREBBE sempre specificare la lunghezza della risorsa completa. Può essere specificato “\*” al posto della lunghezza se questa non è conosciuta al momento dell'invio dell'intervallo di byte. Se un server risponde con status code 206 (Partial content) DEVE sempre essere specificato un intervallo di byte. Se, invece, viene inviato uno status code 406 (Requested range not satisfiable) al posto del range PUO' essere specificato il carattere “\*”.

Quando la risposta contiene un singolo intervallo di byte, è necessario inserire anche l'intestazione “Content-Length” con la lunghezza dell'entity-body parziale. Se la risposta contiene intervalli multipli NON DEVE essere usato il “Content-Range” ma un header “Content-type:” con il tipo MIME “multipart/byteranges “. Questo fa comprendere al client che il messaggio è multipart.

ESEMPLI:

Uso generico, con risorsa completa di lunghezza 1234 byte:

Content-Range: bytes 0-499/1234 → i primi 500 byte

Content-Range: bytes 500-999/1234 → i secondi 500 byte

Content-Range: bytes 500-1233/1234 → tutto esclusi i primi 500 byte

Content-Range: bytes 734-1233/1234 → gli ultimi 500 byte

Uso con un singolo intervallo:

Content-Range: bytes 21010-47021/47022

Content-Length: 26012

Esempio di messaggio multipart: (“boundary” specifica la stringa che separa un messaggio dall'altro)

```
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-type: multipart/byteranges; boundary=THIS_STRING_SEPARATES
```

```
--THIS_STRING_SEPARATES
Content-type: application/pdf
Content-range: bytes 500-999/8000

<payload1>
--THIS_STRING_SEPARATES
Content-type: application/pdf
Content-range: bytes 7000-7999/8000

<payload2>
--THIS_STRING_SEPARATES--
```

<payload> rappresenta i byte del range specificato (il contenuto informativo). Le righe lasciate vuote devono essere rispettate. La stringa finale rappresentante la fine del response differisce dalle altre per la presenza di due caratteri "--" aggiuntivi.

## Content-Type:

E' una intestazione usata sia nelle request che nei response per specificare il MIME type della risorsa che si sta inviando (nelle request è usato solo con il metodo POST). Nel caso il metodo specificato nella request sia un "HEAD", il server invia ugualmente tale intestazione e, in questo caso, essa indica il tipo MIME del entity-body che sarebbe stato inviato se il metodo fosse stato un GET invece di un HEAD.

Attraverso questa intestazione è anche possibile specificare il tipo di codifica dei caratteri (charset) con cui la risorsa inviata deve essere interpretata. Se non è specificato nessun charset, per default il ricevente assume che sia *ISO-8859-1*. Altri tipi di charset sono definiti nell'*RFC1700* (voce Character set). Per siti web che hanno necessità di implementare dei linguaggi non occidentali, il charset più usato è UTF-8 il quale include al suo interno un subset di caratteri ASCII (ed è quindi compatibile con questi ultimi).

ESEMPIO

Content-Type: text/html; charset=utf-8

## Content-Encoding:

Specifica il tipo di codifica che caratterizza la risorsa inviata come entity body e, quindi, in che modo è necessario decodificare quest'ultima per ottenere il tipo di dato specificato nell'header "Content-Type". Va sempre specificato se l'entity body non è di tipo "identity" (vedi Accept-Encoding header). La codifica specificata in questo header è una caratteristica della risorsa identificata dal Request-URL che risulta memorizzata sul server già con quel tipo di codifica.

Se il server riceve una request con specificata una codifica che il server non può soddisfare, questo deve rispondere con un errore 415 (Unsupported Media Type).

ESEMPIO

Content-Encoding: gzip

## Transfer-Encoding:

E' un header che permette di specificare il tipo di trasformazione o codifica che il server ha applicato sull'entity body in modo tale da poterlo trasferire in sicurezza. E' bene sottolineare la differenza che sussiste tra questa intestazione e l'header "Content-Encoding". Quest'ultimo si riferisce al tipo di codifica intrinseca dell'ENTITY BODY, cioè il server non applica alcun tipo di codifica ma semplicemente comunica che quell'entity body è già di suo codificato in quel modo, mentre il "Transfer-Encoding" comunica che è il server ad applicare una codifica (è una proprietà del messaggio e non della risorsa). Se sono state applicate più codifiche è necessario che vengano elencate nell'ordine con cui sono state applicate. E' un header di tipo hop-by-hop e può, quindi, essere applicato anche dagli interlocutori intermedi della catena client/response (ad esempio un entity body può partire in chiaro e poi essere codificato successivamente da un'altra applicazione che non sia il server). Questo header non è compatibile con HTTP 1.0 e non deve essere utilizzato in messaggi afferenti questa versione.

Il Transfer-Encoding è usato principalmente in tutti i quei casi in cui il server non può determinare a priori la lunghezza del messaggio (ad esempio quando abbiamo a che fare con pagine dinamiche la cui lunghezza si determina solo dopo

che il server l'ha elaborata). Quando è presente **Transfer-Encoding, Content-Length NON DEVE essere presente.**

I valori di codifica accettati sono gli stessi dell'header "Accept-Encoding" ai quali si aggiunge il valore "chunked". Questo tipo di codifica prevede di suddividere la risorsa da inviare in chunk, ovvero in pezzi di dati. La lunghezza di questi chunk viene comunicata al client immediatamente prima dell'invio del chunk stesso e la trasmissione termina quando il client riceve un chunk di lunghezza zero. Il valore "chunked" DEVE essere presente sempre quando si utilizza una codifica diversa da "identity" (cioè in chiaro) e la connessione non deve essere terminata con il messaggio corrente. Se sono presenti sia "Content-Encoding" che questo header con un valore di codifica diverso da identity, significa che l'entity body è di suo già codificato secondo il valore del Content e viene nuovamente codificato secondo le modalità specificate nel Transfer. Quindi chi riceve il messaggio deve prima applicare la decodifica dettata dal Transfer e poi quella del Content per ottenere il file "in chiaro". In tutti i casi, se l'intestazione specifica il valore "chunked" questa codifica deve essere applicata sempre per ultima e solo una volta!!!

In ultimo è bene notare che il valore "chunked" è sempre considerato accettabile per interlocutori compatibili con HTTP1.1 anche se non è esplicitamente specificato nell'header "TE" della request. Se quest'ultimo specifica il valore "trailers", il server può inviare delle intestazioni aggiuntive, chiamate "Trailer", dopo l'ultimo chunk della trasmissione. Questi devono essere specificati nell'intestazione "Trailer" del response del server in modo tale che il client sappia che dopo l'ultimo chunk di lunghezza zero ci sono ancora altre informazioni da recuperare. Un trailer è sostanzialmente una intestazione HTTP tra quelle già viste in questo documento, ad eccezione di "Transfer-Encoding", "Content-Length" e "Trailer" che non possono essere usate come Trailer.

ESEMPIO

Transfer-Encoding: chunked

### **Content-Length:**

Indica la grandezza dell'entity body in byte, nel caso del metodo HEAD inviato da un client, la lunghezza dell'entity body che si sarebbe inviato con un metodo GET al posto di HEAD. Va sempre inviato se in un messaggio è presente un entity-body e se è possibile determinare la lunghezza di questo a priori. In caso contrario, si usa il transfer encoding.

ESEMPIO

Content-Length: 348

### **Server:**

Specifica il nome del web server (il software) che ha inviato il messaggio di response. Se il messaggio è inoltrato da un proxy, questo non deve modificare l'header server ma eventualmente aggiungere l'header "Via" con le proprie informazioni.

ESEMPIO

Server: Apache/2.4.1 (Unix)

### **Age:**

E' inserito da un server che dispone di una propria cache ed indica da quanto tempo, in secondi, quella risorsa è presente nella sua cache.

ESEMPIO

Age: 12

### **Last-Modified:**

Specifica la data e l'ora, nel formato HTTP-date, dell'ultima modifica effettuata sulla risorsa richiesta. Dovrebbe essere sempre presente quando questa data è ottenibile.

ESEMPIO:

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

## Etag:

Permette di specificare l'entity tag della risorsa richiesta. Entity tag è un identificatore che permette di individuare una specifica versione di una risorsa. Tale tag può essere esplicitato come validatore “forte”, ovvero che ad ogni cambiamento del suo valore corrisponde una effettiva modifica della risorsa, oppure “debole” (weak) cioè quando ad un cambiamento sulla risorsa ritenuto dal server “di minima rilevanza” non corrisponde un cambiamento del tag.

Permette di implementare molti meccanismi di caching che sono basati proprio sul suo valore.

### ESEMPIO

ETag: "737060cd8c284d8af7ad3082f209582d"

ETag: "xyzy" → validatore forte

ETag: W/"xyzy" → validatore debole

## Vary:

E' utilizzata nei meccanismi di caching. In particolare, è possibile che ad un certo URL possano corrispondere più versioni di una stessa risorsa in cache. Quando una risorsa è in cache, questa intestazione permette di specificare quali valori di determinate intestazioni di una request sono stati utilizzati per recuperare dalla cache quella particolare risorsa. Supponiamo che ci siano due client che vogliono accedere ad una risorsa ma uno dei due accetta la compressione dei dati e l'altro no. Questo significa che i due client nelle loro request utilizzano una intestazione “Accept-Encoding” con valori diversi. In questo caso, quando il server risponde con una risorsa recuperata dalla sua cache specifica una intestazione vary del tipo:

Vary: Accept-Encoding

la quale indica il fatto che la risorsa fornita in risposta è recuperata da una cache ed è stata selezionata in funzione del valore assunto dall'header della response “Accept-Encoding”. Se viene specificato il valore “\*” significa che è stato utilizzato un criterio di selezione della versione della risorsa che non riguarda le intestazioni della request (ad esempio la scelta è stata fatta in funzione dell'indirizzo IP del client, valore non presente nelle intestazioni).

## X-Cache:

E' una intestazione non standard che permette di specificare che il server ha in cache una copia della risorsa richiesta. In tale intestazione si può specificare se la risposta è effettivamente stata restituita dalla cache del server oppure no.

Vediamo i seguenti esempi:

X-Cache HIT from proxy.domain.tld, MISS from proxy.local

In questo caso, il server proxy.domain.tld ci sta dicendo che ha in cache quella risorsa e la risposta è proprio quella della sua cache, mentre il proxy.local ci dice che ha la risorsa in cache ma che non è stata recuperata.

X-Cache MISS from proxy.domain.tld, MISS from proxy.local

In questo caso, entrambi hanno la risorsa in cache ma nessuno dei due l'ha restituita nella risposta poiché quella risposta in realtà proviene dalla cache del browser che ha fatto la request.

X-Cache: HIT from wikipedia.org

In questo caso, abbiamo che la response proviene dalla cache lato server del server web di wikipedia.

## X-Powered-By:

E' una intestazione non standard comunemente utilizzata nelle response di server che supportano determinate tecnologie per l'elaborazione di pagine web dinamiche. Solitamente i server Microsoft IIS utilizzano il valore “ASP.NET”, Apache il valore “PHP/x.x.x”. Non è definito in nessun RFC.

### ESEMPIO

X-Powered-By: PHP/5.4.0