Hypertext Transfer Protocol
Internet Draft
Expires 5 May 1994

Tim Berners-Lee, CERN
draft-ietf-iiir-http-00.ps
5 Nov 1993

Hypertext Transfer Protocol

A Stateless Search, Retrieve and Manipulation Protocol

Status of this memo

This document is an Internet Draft. Internet Drafts are working documents of the Internet Engineering Task Force (IETF), its Areas, and its Working Groups. Note that other groups may also distribute working documents as Internet Drafts.

Internet Drafts are working documents valid for a maximum of six months. Internet Drafts may be updated, replaced, or obsoleted by other documents at any time. It is not appropriate to use Internet Drafts as reference material or to cite them other than as a "working draft" or "work in progress".

This document is a DRAFT specification of a protocol in use on the internet and to be proposed as an Internet standard. Discussion of this protocol takes place on the www-talk@info.cern.ch mailing list –– to subscribe mail to www-talk-request@info.cern.ch. Distribution of this memo is unlimited.

Abstract

HTTP is a protocol with the lightness and speed necessary for a distributed collaborative hypermedia information system. It is a generic stateless object-oriented protocol, which may be used for many similar tasks such as name servers, and distributed object-oriented systems, by extending the commands, or "methods", used. A feature if HTTP is the negotiation of data representation, allowing systems to be built independently of the development of new advanced representations.

Note: This specification

This HTTP protocol is an upgrade on the original protocol as implemented in the earliest WWW releases. It is back-compatible with that more limited protocol.

This specification includes the following parts:

- The Request
- Methods
- A list of headers in the request message
- The response
- Status codes
- Metainformation headers on any object transmitted
- The content of any object content transmitted
- Format negotiation algorithm

- The HTTP Registration Authority
- Security Considerations
- Unresolved points
- References

The following notes form recommended practice not part of the specification:

Servers tolerating clients

Clients tolerating servers

Purpose

When many sources of networked information are available to a reader, and when a discipline of reference between different sources exists, it is possible to rapidly follow references between units of information which are provided at different remote locations. As response times should ideally be of the order of 100ms in, for example, a hypertext jump, this requires a fast, stateless, information retrieval protocol.

Practical information systems require more functionality than simple retrieval, including search, front-end update and annotation. This protocol allows an open-ended set of methods to be used. It builds on the discipline of reference provided by the Universal Resource Identifier (URI) as a name (URN, RFCxxxx) or address (URL, RFCxxxx) allows the object of the method to be specified.

Reference is made to the Multipurpose Internet Mail Extensions (MIME, RFC1341) which are used to allow objects to be transmitted in an open variety of representations.

Overall operation

On the internet, the communication takes place over a TCP/IP connection. This does not preclude this protocol being implemented over any other protocol on the internet or other networks. In these cases, the mapping of the HTTP request and response structures onto the transport data units of the protocol in question is outside the scope of this specification. It should not however be at all complicated.

The protocol is basically stateless, a transaction consisting of

Connection The establishment of a connection by the client to the server - when using TCP/IP port 80 is the well-known port, but other non-reserverd ports may be specified in the URL;

Request The sending, by the client, of a request message to the server;

Response The sending, by the server, of a response to the client;

Close The closing of the connection by either both parties.

The format of the request and response parts is defined in this specification. Whilst header information defined in this specification is sent in ISO Latin-1 character set in CRLF terminated lines, object transmission in binary is possible.

Character sets

In all cases in HTTP where RFC822 characters are allowed, these may be extended to use the full ISO Latin 1 character set. 8-bit transmission is always used.

Table of Contents

1. Request

The request is sent with a first line containing the method to be applied to the object requested, the identifier of the object, and the protocol version in use, followed by further information encoded in the RFC822 header style. The format of the request is:

```
        Request           =     SimpleRequest | FullRequest

SimpleRequest     = GET <uri> CrLf

        FullRequest       = Method UR ProtocolVersion CrLf
[*<HTRQ Header>]
[<CrLf> <data>]

<Method>   =     <InitialAlpha>

ProtocolVersion   = HTTP/1.0

uri   = <as defined in URL spec>

<HTRQ Header>     =     <Fieldname> : <Value> <CrLf>

<data>            =      MIME-conforming-message
```

The URI is the Uniform Resource Locator (URL) as defined in the specification, or may be (when it is defined) a Uniform Resource Name (URN) when a specification for this is settled, for servers which support URN resolution.

Unless the server is being used as a gateway, a partial URL shall be given with the assuptions of the protocol (HTTP:) and server (the server) being obvious.

The URI should be encoded using the escaping scheme described in the URL specification to a level such that (at least) spaces and control characters (decimal 0-31 and 128-159) do not appear unesacaped.

Note. The rest of an HTTP url after the host name and optional port number is completely opaque to the client: The client may make no deductions about the object from its URL.

1.1 Protocol Version

The Protocol/Version field defines the format of the rest of the request.. At the moment only HTRQ is defined .

If the protocol version is not specified, the server assumes that the browser uses HTTP version 0.9.

1.2 Uniform Resource Identifier

This is a string identifying the object. It contains no blanks. It may be a Uniform Resource Locator [URL]defining the address of an object as described in RFCxxxx, or it may be a representation of the name of an object (URN, Universal Resource Name) where that object has been registered in some name space. At the time of writing, no suitable naming system exists, but this protocol will accept such names so long as they are distinguishable from the existing URL name spaces.

1.3 Methods

Method field indicates the method to be performed on the object identified by the URL. More details are with the list of method names below .

1.4 Request Headers

These are RFC822 format headers with special field names given in the list below , as well as any other HTTP object headers or MIME headers.

1.5 Object Body

The content of an object is sent (depending on the method) with the request and/or the reply.

1.6 Methods

The Method field in HTTP indicates the method to be performed on the object identified by the URL. The method GET below is always supported, The list of other methods acceptable by the object are returned in response to either of these two requests.

This list may be extended from time to time by a process of registration with the design authority. Method names are case sensitive. Currently specified methods are as follows:

GET means retrieve whatever data is identified by the URI, so where the URI refers to a data-producing process, or a script which can be run by such a process, it is this data which will be returned, and not the source text of the script or process. Also used for searches .

HEAD is the same as GET but returns only HTTP headers and no document body.

CHECKOUT Similar to GET but locks the object against update by other people. The lock may be broken by a higher authority or on timeout: in this case a future CHECKIN will fail. (Phase out?)

SHOWMETHOD Returns a description (perhaps a form) for a given method when applied to the given object. The method name is specified in a For-Method: field. (TBS)

PUT specifies that the data in the body section is to be stored under the supplied URL. The URL must already exist. The new contenst of the document are the data part of the request. POST and REPLY should be used for creating new documents.

DELETE Requests that the server delete the information corresponding to the given URL. After a successfull DELETE method, the URL becomes invalid for any future methods.

POST Creates a new object linked to the specified object. The message-id field of the new object may be set by the client or else will be given by the server. A URL will be allocated by the server and returned to the client. The new document is the data part of the request. It is considered to be subordinate to the specified object, in the way that a file is subordinate to a directory containing it, or a news article is subordinate to a newsgroup to which it is posted.

LINK Links an existing object to the specified object.

UNLINK Removes link (or other meta-) information from an object.

CHECKIN Similar to PUT, but releases the lock set on the object. Fails if no lock has been set by CHECKOUT. Suggestion : phase out this (rcs-like) model in favor of the PUT (cvs-like, non-locking) model of code management.

TEXTSEARCH The object may be queried with a text string. The search form of the GET method is used to query the object.

SPACEJUMP The object will accept a query whose terms are the cooridnates of a point within the object. The method is implemented using GET with a derived URL .

SEARCH Proposed only. The index (etc) identified by the URL is to be searched for something matching in some sense the enclosed message. How does the client know what message fromats are acceptable to the server? (Suggestion of Fred Williams)

(Some of these methods require more detailed specification)

Note: case sensitivity of method names

Although lack of case sensitivity in methos names would be a tolerant approach with a limited method set, we require the extensibility of HTTP to cover an arbitrary underlying object system. In such a system, method names may be case sensitive, and so we must preserve case in HTTP.

1.6.1 GET

A representation of the object is transferred to the client.

Some URIs refer to specific variants of an object, and some refer to objects with many variants. In the latter case, the representations, encodings, and languages acceptable may be specified in the header request fields, and may affect the particular value which is returned.

Other possible replies allow a set of URIs to be returned to the client, who may use them to retrieve the object. This allows name servers to be implemented using HTTP, and also forwarding address to be given when objects have been moved.

Annotation replies

Some servers keep 'parallel webs" —– separate stores of information about other server's objects. Typically this incldues annotation links. This information may be retrieved using the GET method, but a special reply is returned.

Searching using GET

When the TEXTSEARCH or SPACEJUMP methods are supported, their implementation is in fact by means of the GET method, by constructing a derived URL.

The URL used with GET is the URL of the object, suffixed by a "?" character and the text to be searched. If the object being searched is in fact itself the result of a search (ie the URL contains a "?"), then those search terms are first stripped off, so the search is performed on the original object.

In the URL, keywords are seperated with plus signs ("+"). Real spaces, plus signs, and other illegal characters [who defines illegal characters?]are represented as hexadecimal ASCII escape sequences (%##, e.g., space = %20, plus = %2B)

Browsers accepting text input should define keywords as seperated by spaces (and therefore map them to plus signs). More advanced browsers may allow separate keyword input and, therefore, a finer level of control over the content without users needing to understand the underlying mechanisms (e.g., that they must use %20 to get a real space).

Examples The following is a request on an object supporting TEXTSEARCH:

```
GET   /indexes/botany?annual+plants     HTTP/1.0
```

This request on an object supporting SPACEJUMP selects point (0.2,3.8) within a map:

```
GET   /maps/uk/dorset/winspit/0.2+3.8 HTTP/1.0
```

Note Only one ? operator is allowed in a URL. The convention is that when the result of a search is itself marked as searchable, the search is applied to the original index.

1.6.2 LINK

The link method of HTTP adds meta information (Object header information) to an object, without touching the object's content. For example, it requests the creation of a link from the specified object to another object.

The request is followed by a set of object headers which are to be added.

In cases in which a new header is added and the semantics of the header do not allow it to coexist with a similar header, then the previous header is deleted. For example, an object may only have one title, so specifying a title overwrites and preexsiting title.

Link Types

The link type unless void is specified in the WWW-Link object field for each link. If not specified, void is assumed.

Unresolved points

As this is generalised to allow any metainformation to be added, a better name might be DESCRIBE or attribute (as a verb). I don't like "describe" as it does not suggest alteration. "Bestow"/"Rescind" might be better. For example, one could bestow a title or author on something previously title-less. We are looking at a general data model behind here a little like a relational database. This function adds records. "Set" and "Reset"?

Related Methods

LINK is like POST except that

- no storage is requested for the destination object, and
- the destination is not assumed to be subordinate to the specified object.
- See also: UNLINK.

Tim BL

1.6.3 POST

This method of HTTP creates a new object linked to and subordinate to the specified object. The content of the new object is enclosed as the body of the request.

The POST method is desiged to allow a uniform function to cover

- Annotation existing documents;
- Posting a message to a bulletin board topic, newsgroup, mailing list, or similar group of articles;
- Adding a file to a directory;
- Extending a document during authorship.

The client may not assume any postconditions of the method in terms of web topology. For example, if a POST is accepted, the effect may be delayed or overruled by human moderation, batch processing, etc. The client should not be surprised if a link is not immediately (or never) created.

However, the semantics are a request for a link to be made from the object whose URL is quoted to the new, enclosed, object.

If no URL for the NEW object is given by the client, the server is requested to see to the storage of the new object. That is, the server does not have to store it but will have to return a URL be which it can later be retrieved. The semantics of this method (currently) imply nothing of any undertakings by the server to maintain the availability of the object.

If the client gives a URL, then the server is not obliged to store the object, but may take a copy and may in that case issue a new URL.

Return object headers

The method shall return a set (possibly empty) of object headers for the newly posted object. If a URL has been assigned by the server, then that may be included. Similarly, if a URN has been assigned, then that shall be returned. Other things which may be returned include for example the expiry-date if any. The server may return the entire metadata for the object (as in the HEAD command), or a subset of it.

The object body shall not be returned, so the transaction shall end with the blank line terminating the headers.

Link type

The link type may be specified by explicitly giving a (reverse) link in the object header of the linked object. If a link or links between the two objects are present in those headers, then that link is used. If no such link is specified, then the server should generate a link. The link type in this case is determined by the server. The server may perform other operations as a result of the new object being added: lists and indexes might be updated, for example.

Submission

When articles are submitted, the analogy of being addeed to a body of knowledge by being linked is close. When a form is submitted, this can be done with POST, though in this case side-effects will be expected.

(Should submission for action have a different method –– see showmethod –– or should it be just POST for simplicity? When interfacing with other systems such as bews and mail, the distinction is not made as the system does not have the ability to distinguish different methods. We now have a possibility of making a separate action, though.)

Unresolved points

The client has no way of knowing what data formats the server is prepared to accept. This may not be a serious problem, but it may be if the server has some restrictions. This applies to all submissions of object content from client to server, for all methods.

Related methods.

When a new URL has been returned by the server, it may in general (typically, but not necessarily) be usable as the argument of DELETE , GET , PUT , etc, methods.

To make a link between two existing objects, see LINK. Tim BL

1.6.4 SEARCH

Some correspondance about SEARCH:

Q

```
How did the client know that the server could search for tex?  Maybe this server can only s
ach for text and GIF patterns, even though it gave out tex.  How can the client know that a
```

Answer

```
I assume the following has happened:

client requests with ACCEPT: text/x-TeX

If the server, which it should if possible, replies with the doc
requested in ``TeX''.
```

```
Therefore a server can obviously do an ``internal representation'' to
``TeX'' conversion.

Since the server responded in a particular representation (ie TeX)
The client should be able to search in ``TeX'' provide the ``SEARCH''
method is in the required ALLOWED: section of the response.

The server should only respond with ALLOWED: SEARCH ... if it can do a
Tex to ``internal representation'' conversion for searching.
```

Fred Williams fwilliam@ccs.carleton.ca

1.6.5 SHOWMETHOD

When an object can support more operations than are defined in this specification, SHOWMETHOD allows a client to understand the interface to that operation sufficiently to allow the user to perform it interactively.

Required parameter field

For-Method: This field contains only the method name about which the client is inquiring.

Preconditions

The methodname specified in the For-Method field must have been previously issued in a "Allowed:" field returned with the given object.

The client should specify in Accept: fields formats in which it can accept the result. Possibilities are completely machine-readable descriptions of the method, allowing automatic addition to menus, etc; forms in a form language which all a human to give parameters to the method, and a plain human-readable description of the method and its parameters which is useful for information only.

Postcondidtion

SHOWMETHOD returns, if possible, a form in a representation acceptable to the client. This form will contain instructions for ordering the operation, and fields for the parameters.

A suitable language for the form might be HTTP2, but any language may be used which is acceptable to the client.

1.6.6 SPACEJUMP

This method is similar to the TEXTSEARCH method, but instead of the search criterion being a text string, it is a set of coordinates defining a point within the image. The semantics of the operation are not defined here. Typically, the user clicks on a point within the image with a mouse or other pointing device.

Two or more coordinates are supplied, in the order x, y z, t. All coordinates are scaled so that 0 represents the bottom left hand point and 1.0 represents the top right hand point.

The z access direction follows the normal right-hand rule, that is extends toward the viewer when the x and y axes are flat as in the normal two-dimensional representation.

In the case of a time-occupying object, 0 represents the starting instance, and 1.0 represents the finishing instant.

The method is implemented using GET with a derived URL .

1.6.7 TEXTSEARCH

This is a simple form of search. The text is assumed to derive from the requesting user, and is in no special format. The exact algorithm to be applied is not defined in this specification, but techniques such as vocabulary proximity matching between the request data portion and the contents or titles of documents, keyword matching, stemming, and the use of a thesaurus are quite appropriate.

Whilst this method name is given as a flag to specify that the function is available, the search form of the GET method is in fact used to query the object.

1.6.8 UNLINK

This method deletes metainformation about an object. The request contains object geaders which are to be removed. Only headers exactly matching the headers given are removed.

Obviously the operation may be used for unlinking objects. It may also be used for removing other metainformation such as object title, expiry date, etc.

Related methods

See LINK. Tim BL

1.7 HTTP Request fields

These header lines are sent by the client in a HTTP protocol transaction. All lines are RFC822 format headers. The list of headers is terminated by an empty line.

1.7.1 From:

In Internet mail format, this gives the name of the requesting user. This field may be used for logging purposes and an insecure form of access protection. The interpretation of this field is that the request is being performed on behalf of the person given, who accepts responsability for the method performed.

The Internet mail address in this field does not have to correspond to the internet host which issued the request. (For example, when a request is passed through a gateway, then the original issuer's address should be used).

The mail address should, if possible, be a valid mail address, whether or not it is in fact an internet mail address or the internet mail representation of an address on some other mail system.

1.7.2 Accept:

This field contains a semicolon-separated list of representation schemes (Content-Type metainformation values) which will be accepted in the response to this request.

The set given may of course vary from request to request from the same user.

This field may be wrapped onto several lines according to RCFC822, and also more than one occurence of the field is allowed with the signifiance being the same as if all the entries has been in one field. The format of each entry in the list is (/ meaning "or")

```
<field>  =    Accept: <entry> *[ , <entry> ]
        <entry>  =    <content type> *[ ; <param> ]
<param>  =    <attr> = <float>
        <attr>   =    q / mxs / mxb
        <float>  =    <ANSI-C floating point text represntation>
```

See the appendix on the negotiation algorithm as a function and penalty model. Note that a semicolon has a higher precedence than a comma in this syntax, to conform to MIME use.

If no Accept: field is present, then it is assumed that text/plain and text/html are accepted.

Example

```
Accept: text/plain, text/html
Accept: text/x-dvi; q=.8; mxb=100000; mxt=5.0, text/x-c
```

Wildcards

In order to save time, and also allow clients to receive content types of which they may not be aware, an asterisk "*" may be used in place of either the second half of the content-type value, or both halves. This only applies to the Accept: filed, and not to the content-type field of course.

Example

```
Accept:  *.*, q=0.1
Accept:  audio/*, q=0.2
Accept:  audio/basic q=1
```

may be interpreted as "if you have basic audio, send it; otherwise send me some other audio, or failing that, just give me what you've got."

Type parameters

Parameters on the content type are extremely useful for describing resolutions, colour depths, etc. They will allow a client to specify in the Accept: field the resolution of its device. This may allow the server to economise greatly on transmission time by reducing the resultion of an image, for example, and enable a more appropriate custom-designed black and white image to be selected rathther than giving the client a color image to convert into monochrome.

These parameters are to be specified when types are registered.. @@ TBS.

Sugestions include the following. Please feed back any references to existing improved abreviations for these:

dpi	dots per inch: pixels per inch [cm?!]
pxmax	maximum width in pixels (image or video)
pymax	maximum height in pixels
bits	bits per sample (sound) or pixels (graphics)
mchrome	greyscale or black and white (no value)
sps	samples (sound) or frames (video) per second
length	total size of object in bytes [bits?]

1.7.3 Accept-Encoding:

Similar to Accept, but lists the Content-Encoding types which are acceptable in the response.

```
<field>  =    Accept-Encoding: <entry> *[ , <entry> ]
        <entry>  =    <content transfer encoding> *[ , <param> ]
```

Example

```
                Accept-Encoding: x-compress; x-zip
```

1.7.4 Accept-Language:

Similar to Accept, but lists the Language values which are preferable in the response. A response in an unspecifies language is not illegal. See also: Language .

Language coding TBS. (ISO standard xxxx)

1.7.5 User-Agent:

This line if present gives the software program used by the original client. This is for statistical purposes and the tracing of protocol violations. It should be included. The first white space delimited word must be the software product name, with an optional slash and version designator. Other products which form part of the user agent may be put as separate words.

```
<field>   =   User-Agent: <product>+
        <product> =   <word> [/<version>]
        <version> =   <word>
```

Example:

```
UserAgent:  LII-Cello/1.0  libwww/2.5
```

1.7.6 Referer:

This optional header field allows the client to specify, for the server's benefit, the address (URI) of the document (or element within the document) from which the URI in the request was obtained.

This allows a server to generate lists of back-links to documents, for interest, logging, etc. It allows bad links to be traced for maintenance.

If a partial URI is given, then it should be parsed relative to the URI of the object of the request.

Example:

```
Referer: http://info.cern.ch/hypertext/DataSources/Overview.html
```

1.7.7 Authorization:

If this line is present it contains authorization information. The format is To Be Specified (TBS). The format of this field is in extensible form. The first word is a specification of the authorisation system in use.

Proposals have been as follows: (see the specification for current one implemented by AL Sep 1993)

User/Password scheme

```
Authorization:  user  fred:mypassword
```

The scheme name is "user". The second word is a user name (typically derived from a USER environment variable or prompted for), with an optional password separated by a colon (as in the URL syntax for FTP). Without a password, this povides very low level security. With the password, it provides a low-level security as used by unmodified FTP, Telnet, etc.

Kerberos

```
Authorization:  kerberos  kerberosauthenticationsparameters
```

The format of the kerberosauthenticationsparameters is to be specified.

1.7.8 ChargeTo:

This line if present contains account information for the costs of the application of the method requested. The format is TBS. The format of this field must be in extensible form. The first word starts with a specification of the namespace in which the account is . (This is similar to extensible URL definition.) No namespaces are currently defined. Namespaces will be registered with the registration authority .

The format of the rest of the line is a function of the charging system, but it is recommended that this include a maximum cost whose payment is authorized by the client for this transaction, and a cost unit.

1.8 Note: Server tolerance of bad clients

Whilst it is seen appropriate for testing parsers to check full conformance to this specification, it is recommended that operational parsers be tolerant of deviations.

In particular, lines should be regarded as terminated by the Line Feed, and the preceeding Carriage Return character ignored.

Any HTTP Header Field Name which is not recognised should be ignored in operational parsers.

It is recommended that servers use URIs free of "variant" characters whose representation differs in some of the national variant character sets, punctuation characters, and spaces. This will make URIs easier to handle by humans when the need (such as debugging, or transmission through non hypertext systems) arises.

2. Response

The response from the server shall start with the following syntax (See also: note on client tolerance):

```
<status line>   ::=    <http version>  <status code>  <reason line> <CrLf>
<http version>  ::=    3*<digit>
<status code>   ::=    3*<digit>
<digit>         ::=    0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<reason line> ::=   * <printable>
```

<http version> identifies the HyperText Transfer Protocol version being used by the server. For the version described by this document version it is "HTTP/1.0" (without the quotes).

< status code > gives the coded results of the attempt to understand and satisfy the request. A three digit ASCII decimal number.

<reason string> gives an explanation for a human reader, except where noted for particular status codes.

Fields on the status line are delimited by a single blank (parsers should accept any amount of white space). The possible values of the status code are listed below .

2.1 Response headers

The headers on returned objects those RFC822 format headers listed as object headers , as well as any MIME conforming headers, notably the Content-Type field. Note that this specification doesnot define any headers particular to the response which are not also apropriate to any transmission of an object with a request.

2.2 Response data

Additional information may follow, in the format of a MIME message body. The significance of the data depends on the status code.

The Content-Type used for the data may be any Content-Type which the client has expressed its ability to accept, or text/plain, or text/html. That is, one can always assume that the client can handle text/plain and text/html.

2.3 Status codes

The values of the numeric status code to HTTP requests are as follows. The data sections of messages Error, Forward and redirection responses may be used to contain human-readable diagnostic information.

2.3.1 Success 2xx

These codes indicate success. The body section if present is the object returned by the request. It is a MIME format object. It is in MIME format, and may only be in text/plain, text/html or one fo the formats specified as acceptable in the request.

OK 200

The request was fulfilled.

CREATED 201

Following a POST command, this indicates success, but the textual part of the response line indicates the URI by which the newly created document should be known.

Accepted 202

The request has been accepted for processing, but the processing has not been completed. The request may or may not eventually be acted upon, as it may be disallowed when processing actually takes place. there is no facility for status returns from asynchronous operations such as this.

Partial Information 203

When received in the response to a GET command, this indicates that the returned metainformation is not a definitive set of the object from a server with a copy of the object, but is from a private overlaid web. This may include annotation information about the object, for example

2.3.2 Error 4xx, 5xx

The 4xx codes are intended for cases in which the client seems to have erred, and the 5xx codes for the cases in which the server is aware that the server has erred. It is impossible to distinguish these cases in general, so the difference is only informational.

The body section may contain a document describing the error in human readable form. The document is in MIME format, and may only be in text/plain, text/html or one for the formats specified as acceptable in the request.

Bad request 400

The request had bad syntax or was inherently impossible to be satisfied.

Unauthorized 401

The parameter to this message gives a specification of authorization schemes which are acceptable. The client should retry the request with a suitable Authorization header.

PaymentRequired 402

The parameter to this message gives a specification of charging schemes acceptable. The client may retry the request with a suitable ChargeTo header.

Forbidden 403

The request is for something forbidden. Authorization will not help.

Not found 404

The server has not found anything matching the URI given

Internal Error 500

The server encountered an unexpected condition which prevented it from fulfilling the request.

Not implemented 501

The server does not support the facility required.

2.3.3 Redirection 3xx

The codes in this section indicate action to be taken (normally automatically) by the client in order to fulfill the request.

Moved 301

The data requested has been assigned a new URI, the change is permanent. (N.B. this is an optimisation, which must, pragmatically, be included in this definition. Browsers with link editing capabiliy should automatically relink to the new reference, where possible)

The response contains one or more header lines of the form

```
        URI: <url> String CrLf
```

Which specify alternative addresses for the object in question. The String is an optional comment field. If the response is to indicate a set of variants which each correspond to the requested URI, then the multipart/alternative wrapping may be used to distinguish different sets

Found 302

The data requested actually resides under a different URL, however, the redirection may be altered on occasion (when making links to these kinds of document, the browser should default to using the Udi of the redirection document, but have the option of linking to the final document) as for "Forward".

The response format is the same as for Moved .

Method 303

```
Method: <method> <url>
body-section
```

Like the found response, this suggests that the client go try another network address. In this case, a different method may be used too, rather than GET.

The body-section contains the parameters to be used for the method. This allows a document to be a pointer to a complex query operation.

@@TBS in more detail.

The body may be preceded by the following additional fields as listed .

2.4 Object MetaInformation

The header fields given with or in relation to objects in HTTP are as follows. All are optional. These headers specify metainformation: that is, information about the object, not the information which is contained in the object.

There is no reason for limiting these fields to HTTP use, as any other system which requires metainformation is encouraged to use them.

The order of header lines withing the HTTP header has no significance. However, those fields which are not MIME fields should occur before the MIME fields, so that the MIME fields and following form a valid MIME document. This is not mandatory.

Any header fields which are not understood should be ignored.

(TBS in more detail)

2.4.1 Allowed: *Method

Lists the set of requests which the requesting user is allowed to issue for this URL. If this header line is omitted, the default allowed methods are "GET HEAD"

Example of use:

```
Allow: GET HEAD PUT
```

2.4.2 Public: *method

As "Allow" but lists those requests which anyone may use. If omitted, the default is "GET" only.

Example of use:

```
Public: GET HEAD TEXTSEARCH
```

2.4.3 Content-Length: int

Implies that the body is binary and should be read directly from the communications link, without parsing lines, etc. When the data is part of the request, prevents the escaping and de-escaping of the termination sequence .

@@@ This should be part of the MIME header, as it applies to any binary encoded part. Note HTML is the first internet protocol to allow MIME "binary" encoding. In MIME, the use of Content-Length is currently allowed only for external messages.

2.4.4 Content-Type:

As defined in MIME , except where noted here.

Extra non-MIME types

It is reasonable to put strict limits on transfer formats for mail, where there is no guarantee that the receiver will understand a weird format. However, in HTTP one knows that the receiver will be able to receive it because it will have been sent in the Accept: field . There is therefore a lot to be gained from a very complete registry of well-defined types for HTTP which may nevertheless not be recommended for mail. In this case, the content-type list for HTTP may be a superset of the MIME list.

The x- convention for experimental types is of course still available as well.

Multipart types

MIME provides for a number of "multipart" types. These are encapsulations of several body parts in the one message. In HTTP, Multipart types may be returned on the condition that the client has indicated acceptability (using Accept:)of the multipart type and also of the content types of each consitutent body part.

The body parts (unlike in MIME) MAY contain HTTP metainformation header fields which ARE significant.

Multipart/alternative This is normally used in mail to send different content-type variants when the receiver's capabilities are not known. This is not the case with HTTP. Multipart/alternate may however be used to provide meta information of many instances of an object, in the case of a indirection response . This allows, for example, pointers to be returned by a name server to a set of instances of an object.

Multipart/related This is the type to be used when the first body part contains references to other parts which the server wishes to send at the same time. For example, the first part could be an HTML document, and the included bodyparts could be the inline images mentioned within the text.

The body parts may have URI: fields if the body parts have URIs, and so they may be referred to by these URIs in the body-parts. If the body-parts are transient (as in speech insertions in mail messages) then the [propsed]"cid:" URI type may be used to refer to them by content-identifier.

Multipart/mixed This may be used to simply transfer an unrelated unstructured set of objects. Examples are a picture with audio commentary, etc.

Multipart/parallel This may be used as in MIME to indicate simultaneous presentation. [It is the author's belief that this is a trivial case of a compound presentation which in general should be described by a script which would be the first bodypart of a multipart/related document].

2.4.5 Date: date

Creation date of object. (or last modified, and separately have a Created: field?) Format as in RFC850 but GMT MUST BE USED.

2.4.6 Expires: date

Gives the date after which the information given ceases to be valid and should be retrieved again. This allows control of caching mechanisms, and also allows for the periodic refreshing of displays of volatile data. Format as for Date:. This does NOT imply that the original object will cease to exist.

2.4.7 Last-Modified: date

Last time object was modified, i.e. the date of this version if the document is a "living document". Format as for Date:.

2.4.8 Message-id: uri

A unique identifier for the message. As in RFC850 , except that the unlimited lifetime of HTTP objects requires that the Message-ID be unique in all time, not just in two years.

A document may only have one Message-ID.

No two documents, even if different versions of the same live document, may have the same Message-id.

Note: Unlike the URI field, this does not fgive a way of accessing the document, so the Message-Id cannot be used to refer to the document. In the case of NNTP articles, the message-id may in fact be used within the URI for retrieval using NNTP.

2.4.9 URI: 1*uri

This gives a URI with which the object may be found. There is no guarantee that the object can be retrieved using the URI specified. However, it is guaranteed that if an object is successfully retrieved using that URI it will be to a certain given degree the same object as this one.

If the URI is used to refer to a set of variants, then the dimensiosn in which the variants may differ must be given with the "vary" parameter:

```
Syntax URI: <uri>  [ ; vary = dimension [ , dimension ]* ]
dimension content-type | language | version
```

If no "vary" parameters are given, then the URI may not return anything other than the same bit stream as this object.

Multiple occurencies of this field give alternative access names or addresses for the object.

Examples

```
URI:  http://info.cern.ch/pub/www/doc/url6.multi; vary=content-type
```

This indicates that retrieval given the URI will return the same document, never an updated version, but optionally in a different rendition.

```
URI:  http://info.cern.ch/pub/www/doc/url.multi;
    vary=content-type, language, version
```

This indicates that the URI will return the smae document, possibly in a different rendition, possibly updated, and without excluding the provision of translations into different languages.URI: http://info.cern.ch/pub/www/doc/url6.ps vary=content-typeThis indicates that accessing the URI in question will return exactly the same bitstream.

2.4.10 Version:

This is a string defining the version of an evolving object. Its format is currently undefined , and so it should be treated as opaque to the reader, defined by the informatiuon provider. The version field is used to indicate evolution along a single path of a partucular work. It is NOT used to indicate derived works (use a link), translations , or renditions in different representations .

Note: It would be useful to have sufficient semantics to be able to deduce whether one version predated or postdated another. However, it may also be useful to be able to insert a particular local code management system's own version stamp in this field. Typically, publishers will have quite complex version information containing hidden local semantics, giving value to the idea of this field being opaque to other readers ofthe document.

2.4.11 Derived-From:

When an editied object is resubmitted using PUT for example, this field gives the value of the Version . This typically allows a server to check for example that two concurrent modifications by different parties will not be lost, and for example to use established version management techniques to merge both modifications.

2.4.12 Content-Language: code

The language code is the ISO code for the language in which the document is written. If the language is not known, this field should be omitted of course .

The language code is an ISO 3316 language code with an optional ISO639 country code to specify a national variant.

Example

```
 Language: en_UK
```

means that the content of the message is in British English, while

```
  Language: en
```

means that the language is English in one of its forms. (@@ If a document is in more than one language, for example requires both Greek Latin and French to be understood, should this be representable?)

See also: Accept-Language .

2.4.13 Cost: TBS

The cost of retrieving the object is given. This is the cost of access of a copyright work, with a specification of the payment system accepted. Format to be specified. Currently refers to an unspecified charging scheme to be agreed out of band between parties.

Note that ISO 4217 provides a standard for desription of currency codes, but that one can imagine payment methods which have their own currency (Club Med iI believe uses cloured beads).

2.4.14 Link:

Note. It is proposed that any HTML metainformation element (allowed withing the HEAD as opposed to BODY element of the document) be a valid candidate for an HTTP object header. LINK is one example, TITLE another. One suggestion was that the isomorphism should be realized by prepending "WWW-" to the HTML element name to make the HTTP header name, and the HTML attributes imply identically named semicolon-separated MIME-style header parameters. It is open to discussion whether the "WWW-" should be inserted or not.

This is semantically equivalent to the LINK element in an HTML document which should be consulted for a full explanation.

Examples

```
WWW-Link:  href="http://info.cern.ch/a/b/c"; rel="includes"
WWW-Link:  href="mailto:timbl@info.cern.ch"; rev=made
```

The first example indicates that this object includes the specified /a/b/c object. The second indicates that the author of the object is identified by the given mail address.

2.4.15 Title:

The title of the document. Not part of the document. Syntax is as for the Subject field in RFC822. The Title field differs from the Subject field in that the title is defined by the author of a document, but the Subject field is defined by the originator.

WWW note: Isomorphic with the <TITLE> element in HTML .

2.5 Note: Client tolerance of bad servers

Servers not implementing the specification as written are not HTTP compiant. Servers should always be made completely copmpliant. However, clients should also tolerate deviant servers where possible.

2.5.1 Back compatibility

In order that clients using the HTTP protocol should be able to communicate with servers using the protocol originally implemented in the W3 data model, clients should tolerate responses which do not start with a numeric version number and response codes.

In this case, they should assume that the rest of the response is a document body in type text/html.

2.5.2 White space

Clients should be tolerant in parsing response status lines, in particular they should accept any sequence of white space (SP and TAB) characters between fields.

Lines should be regarded as terminated by the Line Feed, and the preceeding Carriage Return character ignored.

3. HTTP Negotiation algoritm

This note defines the significance of the q, mxb and mxs values optionally sent in the Accept: field of the HTTP protocol request message.

It is assumed that there is a certain value of the presentation of the document, optimally rendered using all the information available in its original source.

It is further assumed that one can allocate a number between 0 and 1 to represent the loss of value which occurs when a document is rendered into a representation with loss of information. Whilst this is a very subjective measurement, and in fact largely a function of the document in question, the approximation is made that one can define this "degradation" figure as a function of merely the representation involved.

The next assumption is that the other cost to the user of viewing the document is a function of the time taken for presentation. We first assume that the cost is linear in time, and then assume that the time is linear in the size of the message.

The final net value to the user can therefore be written

presented_value = initial_value * total-degradation - a - b * size

for a document in a given incoming representation. Suppose we normalize the initial value of the document to be 1. The server may judge that the value in a particular format is less than 1 is a conversion on the server side has lost information. The total degradation is then the product of any degradation due to conversions internal to the server, and the degradation "q" sent in the Accept field. If q is not sent, it defaults to 1.

The values of a and b have components from processing time on the server, network delays, and processing time on the client. These delays are not additive as a good system will pipeline the processing, and whilst the result may be linear in message size, calculation of it in advance is not simple. The amount of pipelining and the loads on machines and network are all difficult to predict, so a very rough assumption must be made.

We make the client responsible for taking into account network delays. The client will in fact be in a better position to do this, as the client will after one transaction be aware of the round-trip time.

We assume that the delays imposed by the server and by the client (including network) are additive. We assume that the client's delay is proportional to message size.

The three parameters given by the client to the server are

q The degradation (quality) factor between 0 and 1. If omitted, 1 is assumed.

mxb The size of message (in bytes) which even if immediately available from the server will cause the value to the reader to become zero

mxs The delay (in seconds) which, even for a very small message with no length-related penalty, will cause the value to the reader to become zero.

These parameters are chosen in part because they are easy to visualize as the largest tolerable delay and size. If not sent, they default to infinity.

The server may optimize the presented value for the user when deciding what to return. The hope is that fine decisions will not have to be made, as in most cases the results for different formats will be very different, and there will be a clear winner.

A suitable algorithm is that the assumed value v of a document of initial value u delivered to the network after a delay t whose transfer length on the net is b bytes is

v = u * q - b/mxb - t/mxs

Note that t is the time from the arrival of the request to the first byte being available on the net. [[See also: Design issues discussions around this point.]]

3.1 HTTP Negotiation algoritm

This note defines the significance of the d, a and b values optionally sent in the Accept: field of the HTTP protocol request message.

It is assumed that there is a certain value of the presentation of the document, optimally rendered using all the information available in its original source.

It is further assumed that one can allocate a number between 0 and 1 to represent the loss of value which occurs when a document is rendered intop a represenmtation with loss of information. Whilst this is a very subjective measurement, and in fact largely a function of the document in question, the approximation is made that one can define this "degradation" figure as a function of merely the representation involved.

The next assumption is that there the other cost to the user of viewing the doument is a function of the time taken for presentation. We first assume that the cost is linear in time, and then assume that the time is linear in the size of the message.

The final net value to the user can therefore be written

presented_value = initial_value * total-degradation - a - b * size

for a document in a given incoming represenattion. Suppose we normalize the initial value of the document to be 1. The total degradation is the product of any degradation due to conversions internal to the server, and the degradation "d" sent in the Accept field. The values of a and b are also sent by the protocol. If not sent, they default to no cost (d=1, a=b=0).

The server may optimize the presented value for the user when deciding what to return. The hope is that fine decisions will not have to be made, as in most cases the results for different formats will be very different, and there will be a clear winner.

See also: Design issues discussions around this point. Tim BL

3.2 Note: The cost of retrieval time

The assumption that the cost to the user associated with a certain retrieval time is linear in that time is wildly innaccurate. The real function could be very dependent on circumstances (like go to infinity at a deadline).

A better general approximation might be logarithmic for large time delays, and linear for small ones, like a*log(b*t-1) which has two parameters.

4. Registration Authority

The HTTP Registration Authority is responsible for maintaining lists of:

- Charge account name spaces (see ChargeTo: field above)
- Authorization schemes (see Authorization: field above)
- Data format names (as MIME Content-Types)
- Data encoding names (as MIME Content-Encoding))

It is proposed that the Internet Assigned Numbers Authority or their successors take this role. Unregistered values may be used for experimental purposes if they are start with "X-".

5. Security Considerations

The HTTP protocol allows requests to communication to a remote server machine, and all the expetant security considerations for client-server systems apply, including

- Authentication of requests
- Authenticationtion of servers
- Privacy of request and response
- Abuse of server features
- Abuse of servers by exploiting server bugs
- Unwitting actions on the net
- Abuse of log information

The bulk of these are well known problems, tackled in part by some featured of this protocol. Some aspects particular to HTTP are mentioned below.

5.1 Unwitting actions on the net

The writers of client software should be aware that the software represents the user in his interactions over the net, and should be careful to allow the user to be aware of any actions he may take which may be taken as having an unexpected significance by others.

5.1.1 TCP port numbers

Clients should prompt a user before allowing HTTP access to reserved ports other than the port resrverd for HTTP (port 80). Otherwise, the user may unwittingly cause a transaction to occur in some other (present or future) protocol.

5.1.2 Idempotent methods

The convention should be established that the GET and HEAD methods never have the significance of taking an action. The link "click here to subscribe" causing the reading of a special "magic" document is open to abuse by others making a link "click here to see a pretty picture". These methods should be considered "safe" and should not have side effects. This allows the client software to represent other, methods (such as POST) in a special way, so that the use is aware of the fact that an action is being requested.

5.2 Abuse of log information

A server is in the position to save large amount of personal data about information requested by different readers. This information is clearly confidential in nature, and its handling may be constrained by law in certain countries. Server providers shall ensure that such material is not distributed without the permission of any people or groups of people mentioned in the results published.

A feature which increases the amount of personal data transferred is the Referer: field. This allow reading patters to be studied, reverse links drawn, and so is very useful. Its power can be abused of course if user details are not separated from the Referee-Referer pairs. Even when the personal information has been removed, the Referer field may in fact be a secure document's URI, whose revelation itself is breach of security. A method of suppressing the Referer information in such cases may be the subject of further study.

6. References

RFC 822 "Standard for ARPA Internet Text Messages". David H. Crocker, describes Internet mail message fromat.

RFC850 "Standard for Interchange of USENET Messages" This RFC uses some field names in common with this specification, and is relevant reading.

RFC977 "Network News Transfer Protocol", Kantor and Lampsley.

RFC 1341 Multipurpose Internet Mail Extensions (MIME), Nathaniel Borenstien and Ned Freed, Internet RFC 1341, 1992. Now obsoleted by RFC1521 :

RFC 1521 MIME. Not available in hypertext form yet.

RFC 1522 K Moore. "MIME: Part Two: Message Header Extensions for Non-ASCII Text".

RFC1523 MIME text/enriched.

RFC 1524 MIME...

URL Universal Resource Locators. RFCxxx. Currently available by anonymous FTP from info.cern.ch as /pub/ietf/url3.{ps,txt}.

MIME and PEM Internet Draft only

(end of references)

6.0.1 Object Contents

The data (if any) sent with an HTTP request or reply is in a format and encoding defined by the object header fields, the default being "plain/text" type with "8bit" encoding. Note that while all the other information in the request (just as in the reply) is in ISO Latin1 with lines delimited by Carriage Return/Line Feed pairs, the data may contain 8-bit binary data.

Termination The delimiting of the message is determined by the Content-Length: field. If this is present, then the message contains the specified number of bytes.

Failing that, the content-type filed may contain a "bounday" attribute which gives the boundary string with exacly the same syntax as for a MIME multipart message.

Failing either of the above conditions, the data is terminated by the closing of the connection by the sending party. Note that this method can not be used for data sent with the request.

See also: note on server tolerance for back-compatibility, etc.