HTTP: A protocol for networked information

Generated from the Hypertext

June 2, 1993

Table of Contents

Chapter 1: Protocol for the Retrieval and Manipulation of Textual and Hypermedia Information

1.1 Status of this memo

This document is a DRAFT specification of a protocol in use on the internet and in time 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.

1.2 Note: This specification

The HTTP2 protocol is an upgrade on the HTTP protocol as implemented in the initial WWW release. It is simply a name for an upgrade to the protocol, and is intended to be back-compatible with the old protocol.

The HTTP protocol implements the WWW data model . A prime concern is that servers must be able to skip data they don't understand. This will guarantee forward compatibility.

Other parts of this specification are:

- A list of headers in the request message
- Status codes
- A list of headers on any object transmitted
- Format negotiation algorithm
- The HTTP Registration Authority
- References

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

- Tolerating clients
- tolerating servers

1.3 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.

(TBContinued)

1.4 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;

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 US-ASCII character set in CRLF terminated lines, object transmission in binary is possible.

Chapter 2: 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           = Method Url ProtocolVersion CrLf
[*<HTRQ Header>]
[<CrLf> <data>]

<Method>   =     <InitialAlpha>

ProtocolVersion   = HTRQ/V1.0

URI   = <as defined in URL spec>

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

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

The URI is the Universal Resource Locator as defined in the specification.

2.1 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.

2.2 Universal Resource Identifier

This is a string identifying the object. It contains no blanks. It may be a Universal 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.

2.3 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 .

2.4 Request Headers

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

2.5 Data

The data (if any) sent with an HTTP request is a MIME conforming message body. Note that while all the other information in the request (just as in the reply) is in IISO Latin1 with lines delimited by Carriage Return/Line Feed pairs, the data may contain 8-bit binary data.

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

2.6 Methods

Method field indicates the method to be performed on the object identified by the URL. The methods GET and HEAD below are 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 Udi, so where the Udi 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.

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.

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 Udi.

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.

CHECKIN Similar to PUT, but releases the lock set on the object. Fails if no lock has been set by CHECKOUT.

TEXTSEARCH This is a simple form search taking a text argument which is sent in the data portion of the request. 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.

2.7 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.

2.7.1 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.

2.7.2 Accept:

This field contains a comma-separated list of representation schemes (MIME compatible Content-Type 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.

Example

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

2.7.3 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>
```

Example

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

2.7.4 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
```

2.7.5 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
```

2.7.6 Authorization:

This line is present 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:

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.

2.7.7 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.

2.8 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.

For backward compatibility, a GET method with a URL containing a search operator ("?") should be supported by any server supporting TEXT SEARCH.

Chapter 3: 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 .

3.1 Response headers

The headers on returned objects are RDC822 format headers with special field names given below , as well as any MIME conforming headers, notably the Content-Type field.

3.2 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 his 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.

3.3 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.

3.3.1 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.

3.3.2 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 URL given

Internal Error 500

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

Not implemented 501

The server does not support the facility required.

3.3.3 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

```
Location: <url> String CrLf
```

Which specify alternative addresses for the object in question.

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.

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

3.4 Object Headers

The header fields given with or in relation to objects in HTTP are as follows. All are optional.

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.

(TBS in more detail)

3.4.1 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
```

3.4.2 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
```

3.4.3 Length: int

Implies that the body is binary and should be read directly from the communications link, without parsing lines, etc.

@@@ 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.

3.4.4 Date: date

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

3.4.5 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.

3.4.6 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:.

3.4.7 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.

3.4.8 Version-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 the same unmodified object as this one.

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

3.4.9 Live-URI: 1*uri

This gives a URI with which the most recent version of an 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 that it will be the same object or a more recent version of the same object.

Multiple occurencies of this field give alternatives which should refer to the same live object.

3.4.10 Cost: TBS

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

3.5 Note: Client tolerance of bad servers

3.5.1 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.

3.5.2 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.

Chapter 4: 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.]]

4.1 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.

Chapter 5: 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-".

Chapter 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.
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