INTERNET-DRAFT
Expires: August, 1996

Ari Luotonen
Netscape Communications Corporation
John Franks
Northwestern University

<draft-ietf-http-range-retrieval-00.txt>

February 22, 1996

Byte Range Retrieval Extension to HTTP

STATUS OF THIS MEMO

This document supersedes <draft-luotonen-http-url-byterage-02.txt>.

A subgroup of the HTTP working group has developed this after several rounds of discussion. Some but not all parts of this proposal are currently implemented in commercial Web servers and browsers.

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 draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as ``work in progress.''

To learn the current status of any Internet-Draft, please check the ``1id-abstracts.txt'' listing contained in the Internet-Drafts Shadow Directories on ftp.is.co.za (Africa), nic.nordu.net (Europe), munnari.oz.au (Pacific Rim), ds.internic.net (US East Coast), or ftp.isi.edu (US West Coast).

TABLE OF CONTENTS

1. OVERVIEW

It is possible in Web clients to interrupt the connection before the data transfer has finished. As a result the client may have partial documents or images loaded into its memory. It would be useful to be able to request the server to return the missing portion of the document only, instead of retransferring the entire file, if the page is re-entered later.

There are also a number of Web applications that would benefit from being able to request the server to give a byte range of a document. As an example an Adobe PDF viewer would need to be able to access individual pages by byte range; the table that defines those ranges is located at the end of the PDF file (this is the case in the new to-be-released PDF format).

Setting this standard will promote interoperability between clients, servers and intermediate proxy servers, make (partial) caching effective, and save bandwidth.

This specification defines only the byte ranges. It shows other types of ranges as an example of how this specification could be extended, as proof of its generality. Those examples should not be viewed as their definition.

This specification is simple enough to be adopted quickly by the server authors/vendors, and be quickly and easily exploited on the client side. The proposed solution will be backward compatible with existing proxy servers, and once this specification becomes official it will actually be possible to support this in a smart way in proxy servers.

This specification can be applied to document types for which byte ranges make sense; there are types for which they don't, and this specification is not trying to enforce semantics for byte ranges for

them. In practice most of the data in the Web is represented as a byte stream, and can be addressed with a byte range to retrieve a desired portion of it. This is especially useful when there is a partial copy of the document, the transfer of which was interrupted by the user, but later resumed, in which case only the missing portion needs to be transferred.

Byte range requests are typically generated by software, not written by humans.

2. ACCEPT-RANGES HTTP RESPONSE HEADER

The server needs to let the client know that it can support byte ranges. This is done through the Accept-Ranges HTTP header when a server is returning a document that supports byte ranges:

```
Accept-Ranges: bytes
```

The server will send this header only for documents for which it will be able to satisfy the byte range request, e.g. for PDF documents, or images, which can be partially reloaded if the user interrupts the page load, and image gets only partially cached.

Because of the architecture of the byte range request and response, the client is not limited to attempting to use byte ranges only when this header is present. The Request-Range header is simply ignored by a server that does not support it, and it will send the entire document as a response.

3. BYTE RANGE HTTP REQUEST

Byte range request is made like any other HTTP request, with the addition of the Range: HTTP Request header.

3.1. The Range HTTP Request Header

The client requests a byte range via the Range: HTTP header:

```
Range: bytes=0-500,5000-
```

The Range: header is defined extensibly so that it can take a generic parameter specifying the type of range. The parameter name for byte ranges is "bytes". The syntax of this parameter is described below.

The name of the byte range parameter is bytes. It is passed to the server in the Range: HTTP request header, followed by an equal sign and the byte range specification. (In an earlier version of this draft, it was passed to the server appended to the end of the path part of the URL, separated by a semicolon).

Note About CGI Applications

As defined by the CGI/1.1 specification, the value of the Range: header will be passed to CGI scripts in the HTTP_RANGE environment variable. The CGI applications can choose to support it if they so desire, and if it is possible. If the CGI applications do not support it, or if the content they return changes from call to call, they simply ignore the presence of that header, and return the entire document.

Each range consists of one or two non-negative integers, separated by a hyphen. The first integer must always be less than or equal to the second one. One of these integers may be missing, but not both at the same time. The hyphen is always there, so it is possible to tell which number is missing.

If the first number is missing, it means to return the n last bytes of the document, where n is the second number. If n is equal to, or larger than, the size of the document minus one, then the entire file is returned.

If the second number is missing, it means the end of document. That is, all the bytes starting from byte n until the end of the document, where n is the first number.

The first byte in a document is byte number 0.

If the second number is larger than the size of the document minus one, it is taken to mean the size of the document minus one (that is, the end of the document).

The range is inclusive; as an example, the range 500-1000 includes bytes from 500 to 1000, including 500 and 1000.

There may be multiple ranges, separated by a comma. The order of the ranges is the preferred order in which the ranges should be returned.

In the case that the second integer is smaller than the first one, that particular range is tagged as invalid, and ignored. If it was the only requested byte range, the entire document is returned.

Otherwise the remaining valid ranges will be returned.

The byte ranges refer to ranges in data as they are transferred over the network (and retrieved by the client). E.g. if in an imaginary system the server stores all lines terminated by CR LF, but turns them into a single LF before sending the data, then byte ranges refer to ranges inside this modified data (the one with single LF line separators). That is, the ranges refer to the data that the client would see.

The byte ranges apply to the "raw" data, that is, the data encoded by Content-encoding; but not to the "armored" data, that is, the data encoded by content-transfer-encoding.

Examples of the Range: header with the bytes parameter

The first 500 bytes:

```
Range: bytes=0-499
```

The second 500 bytes:

```
Range: bytes=500-999
```

All bytes except for the first 500 until the end of document:

```
Range: bytes=500-
```

The last 500 bytes of the document:

```
Range: bytes=-500
```

Two separate ranges:

```
Range: bytes=50-99,200-249
```

The first 100 bytes, 1000 bytes starting from the byte number 500, and the remainder of the document starting from byte number 4000 (byte numbering starts from zero):

```
Range: bytes=0-99,500-1499,4000-
```

The first 100 bytes, 1000 bytes starting from the byte number 500, and the last 200 bytes of the document:

```
Range: bytes=0-99,500-1499,-200
```

3.2 Conditional Range Retrievals

There are three cases of Range retrievals:

A) Unconditional
B) Insertion
C) Tentative

A) Unconditional Range Retrieval

An unconditional range retrieval always returns the selected range (if it exists).

Servers that do not support Range: return entire resource.

B) Insertion Range Retrieval

Insertion type range retrieval returns the selected range if the request-validator is valid, otherwise returns entire resource.

Servers that do not support Range: return entire resource if the validator is valid, else return nothing + "306 Modified".

Case (B) is for filling in a hole in a cached resource, perhaps after an interrupted retrieval, or perhaps after a previous Unconditional or Tentative Range: request.

Note that if the server does not support Range:, it requires an extra round-trip to update a cached resource with a hole in it.

C) Tentative Range Retrieval

A tentative range retrieval returns the selected range if the request-validator is NOT valid, otherwise returns nothing + "304 Not Modified"

Servers that do not support Range: return entire resource if the validator is not valid, else return nothing + "304 Not Modified".

Case (C) is a way of asking only for a specified range, but in a way that reloads it only if the client's cache is stale. For example, you may have a large GIF image in your cache, and you may want to know if its GIF header has changed (so that you can do an early rendering of the enclosing HTML file), but you do not want to retrieve the whole image right away.

Examples

The three cases are implemented in HTTP/1.1 using some combination of these three headers

```
        Range:
        If-invalid:
        If-valid:
```

or

```
        Range:
        If-Modified-Since:
        Unless-Modified-Since:
```

See (*) below for more about using *-Modified-Since.

Case A (Unconditional) is done using

```
        GET /image.gif HTTP/1.0
        Range: bytes=0-128
```

Case B (Insertion) is done using

```
        GET /image.gif HTTP/1.0
        Range: bytes=0-128
        If-valid: "xxxyxxyx"
```

or

```
        GET /image.gif HTTP/1.0
        Range: bytes=0-128
        Unless-modified-since: Wed, 15 Nov 1995 06:25:15 GMT
```

Case C (Tentative) is done using

```
        GET /image.gif HTTP/1.0
        Range: bytes=0-128
        If-invalid: "xxxyxxyx"
```

or

```
        GET /image.gif HTTP/1.0
        Range: bytes=0-128
        If-modified-since: Wed, 15 Nov 1995 06:25:15 GMT
```

(*) We can get rid of "Unless-modified-since:" if we adopt the rule that the "If-valid:" header either takes a quoted string (in which case it is an opaque validator) or a string that does not include quotes (in which case it must be an HTTP-date value). This saves a few header bytes.

4. BYTE RANGE HTTP RESPONSE

4.1. 206 Partial Content Status Code

The byte range response uses the 206 Partial Content HTTP response status. Servers and CGI applications not supporting byte ranges will simply ignore the Range: header in the request, and return the entire document in a 200 response.

Existing proxy servers only cache 200 Ok responses. This way intermediate proxy servers will not mistakenly cache a partial document as if it was the entire document.

If the request includes multiple ranges, the response is sent back as a multipart MIME message, with content-type multipart/x-byteranges. A server may, but is not required to, send also a single byte range as a multipart message.

If there are overlapping ranges the behavior for each range doesn't change. That is, a range will not be truncated, merged, or left out, just because there is an overlap.

If a request that includes a Range: header also includes an Unless-Modified-Since header and the resource was modified since the specified time, or also includes an If-Valid header and the validator does not match,

- if the server supports Range:, it will send a normal 200 OK response, and transfer the entire resource instead.

- if the server does not support Range:, it will send 306 Modified response, and will not transfer any part of the resource.

- if the server supports neither Range: nor the validation header (Unless-Modified-Since or If-Valid) then it will send a normal 200 OK response, and transfer the entire resource.

If a request that includes a Range: header also includes an If-Modified-Since header and the resource was not modified since the specified time, or also includes an If-Invalid header and the validator matches, the server will send a 304 Not Modified response, and will not transfer any part of the resource.

4.2 The Content-Range HTTP Response Header

The Content-Range HTTP response header is sent back to provide verification and information about the range and total size of the document. This header can be used by the client to determine which one of the requested ranges is in question. Syntax:

```
Content-Range: bytes X-Y/Z
```

where:

X is the number of the first byte returned (the first byte is byte number zero).

Y is the number of the last byte returned (in case of the end of the document this is one smaller than the size of the document in bytes).

Z is the total size of the document in bytes.

Examples of the Content-Range: HTTP Response Header

The first 500 bytes of a 1234 byte document:

```
Content-Range: bytes 0-499/1234
```

The second 500 bytes of the same document:

```
Content-Range: bytes 500-999/1234
```

All bytes until the end of document, except for the first 500 bytes:

```
Content-Range: bytes 500-1233/1234
```

The last 500 bytes of the same document:

```
Content-Range: bytes 734-1233/1234
```

Example of a response:

```
HTTP/1.0 206 Partial content
Server: Netscape-Communications/2.0
Date: Wed, 15 Nov 1995 06:25:24 GMT
Last-modified: Wed, 15 Nov 1995 04:58:08 GMT
Content-range: 21010-47021/47022
Content-length: 26012
Content-type: image/gif
```

4.3. Multiple Ranges as Multipart MIME Messages

Multipart MIME is defined in [RFC-1521]. With byteranges, the multipart MIME message uses content-type multipart/x-byteranges, with a boundary parameter.

Example:

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

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

...the first range...
--THIS_STRING_SEPARATES
Content-type: application/x-pdf
Content-range: bytes 7000-7999/8000

...the second range...
--THIS_STRING_SEPARATES--
```

4.4. Caching Issues

The server must give Last-modified headers for each range request whenever possible, and the client side must take care of having all the fragments in sync. Conditional GET (the GET request with the If-modified-since header) works as expected with byte ranges. That is, the requested range is returned if the document has been modified since the given date. Otherwise, a 304 Not Modified response is sent.

Ranges can be cached, and if the Last-modified header matches they can be combined. If a received Last-modified date at any time differs from the ones in the cache, all the cached ranges will be discarded.

The client side should monitor the Last-modified header value returned by the server, and make sure that all of its individual fragments are in sync. If there are older ones they should be immediately discarded and re-retrieved.

5. FUTURE CONSIDERATIONS

5.1. Extending Accept-Ranges, Range and Content-Range headers

If at some point there will be additional parameters for the Range: header, they should be separated by the semicolon character.

Example:

```
Range: param1=bar; param2=xyzzy
```

This specification does not define semantics for cases with multiple Range: parameters. Future specifications should define semantics for these. Until then, Range: headers with parameters that cannot be understood should be ignored.

5.2. Other Possible Ranges

There are other kinds of ranges that can be addressed in a similar fashion; this document does not define them, but both the Range: HTTP request header and the Content-Range: HTTP header are defined so that it is possible to extend them.

As an example, there might be a "lines" parameter, with the same kind of range specification, and the Content-Range: header would then specify the numbers in lines. Example:

```
GET /dir/foo HTTP/1.0
Range: lines=20-30
```

The response from a 123 line document would be:

```
HTTP/1.0 206 Partial Content
Content-Range: lines 20-30/123
Last-Modified: ...
Content-Length: 773
Content-Type: text/plain
```

This could be useful for such things as structured text files like address lists or digests of mail and news, but isn't meaningful to such document types as GIF or PDF.

Other examples might be document format specific ranges, such as chapters:

```
GET /dir/foo HTTP/1.0
Range: chapters=6-9

206 Partial Content
Content-Range: chapters 6-9/12
Last-Modified: ...
Content-Length: 36023
Content-Type: application/x-book-type
```

6. References

[RFC-1521] N. Borenstein, N. Freed, "MIME (Multipurpose Internet Mail Extensions), Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies", RFC 1521, Bellcore, Innosoft, September 1993

[HTTP] T. Berners-Lee, R. Fielding, H. Frystyk, "Hypertext Transfer Protocol -- HTTP/1.0", draft-ietf-http-v10-spec-04.html, October 14, 1995.

[CGI] R. McCool et al, "Common Gateway Interface -- CGI/1.1", http://hoohoo.ncsa.uiuc.edu/cgi/, NCSA, 1994.

7. Authors' Addresses:

Ari Luotonen <ari@netscape.com>
Netscape Communications Corporation
501 E. Middlefield Road
Mountain View, CA 94043
USA

John Franks <john@math.nwu.edu>
Department of Mathematics
Northwestern University
Evanston, IL 60208-2730

The subgroup of the HTTP working group that contributed in producing this draft (in alphabetical order; apologies for any omissions):

Daniel W. Connolly	connolly@beach.w3.org
Roy T. Fielding	fielding@avron.ics.uci.edu
John Franks	john@math.nwu.edu
Jim Gettys	jg@w3.org
Martin Hamilton	martin@mrrl.lut.ac.uk
Koen Holtman	koen@win.tue.nl
Shel Kaplan	sjk@amazon.com
Paul Leach	paulle@microsoft.com
Alex Lopez-Ortiz	alopez-o@barrow.uwaterloo.ca
Ari Luotonen	luotonen@netscape.com
Larry Masinter	masinter@parc.xerox.com
Jeff Mogul	mogul@pa.dec.com
Lou Montulli	montulli@mozilla.com
David W. Morris	dwm@shell.portal.com
Luigi Rizzo	luigi@labinfo.iet.unipi.it
Bill Weihl	weihl@pa.dec.com
Steve Zilles	szilles@mv.us.adobe.com