Progress notifications for HTTP
draft-decroy-http-progress-00

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with Section 6 of BCP 79.

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."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on August 1, 2007.

Copyright Notice

Copyright (C) The Internet Society (2007).
Abstract

This document specifies extensions to HTTP to allow progress messages for user-agents during lengthy transactions, and to allow better flow-control of large message body submission in cases where HTTP authentication is required by servers and/or intermediaries.

Table of Contents

1. Terminology .................................................. 3
2. Introduction and motivation .................................. 4
  2.1 The need for progress information. ....................... 4
  2.2 Flow control issues. ...................................... 4
  2.3 Aim of this document ..................................... 4
  3.1 100 Continue .............................................. 5
    3.1.1 Further discussion of the flow-control problem .... 5
    3.1.2 Use of 100 continue for progress messages ......... 6
  3.2 102 Defer Transmission .................................. 7
4. Header Field Definitions ................................... 8
  4.1 Progress .................................................. 8
    4.1.1 Client request ....................................... 8
    4.1.2 1xx Responses ........................................ 9
5. Examples ..................................................... 10
  5.1 Use of 102 Defer and 100 Continue for flow control .... 10
6. Compatibility issues. ....................................... 12
  6.1 Client does not support this specification and server does 12
  6.1 Client supports this specification and server does not ... 12
7. Implementation Notes ......................................... 13
8. Security Considerations ..................................... 14
9. Notes & TODO ................................................ 15
10. IANA Considerations ........................................ 16
11. References .................................................. 17
  11.1. Normative References ................................. 17
Author’s Address ............................................... 18
Intellectual Property and Copyright Statements ................ 19
1. Terminology

The keywords "MUST", "MUST NOT", "REQUIRED", "SHOULD", "SHOULD NOT" and "MAY" that appear in this document are to be interpreted as described in [RFC2119]
2. Introduction and motivation

2.1 The need for progress information

Increasingly, resource transfers using HTTP are subjected to ever more complex processing, particularly by proxy servers. Processing such as scanning resources for viruses at an HTTP proxy creates special problems that cannot be resolved cleanly with the current specification of HTTP.

Many types of processing require access to an entire message body. This can take considerable time to accumulate depending on link bandwidth etc. A common example of the problem occurs when scanning content at a proxy gateway for viruses. The scanning software requires access to the entire resource, however for large resources on slow links this can take considerable time to accumulate then scan before any response can safely be sent to the user agent.

Frequently problems occur where users of client agents give up waiting for visible progress. Several methods of dealing with the problem have been deployed, each with associated other problems.

There is a clear need for upstream agents to be able to provide timely progress notifications to downstream users to enable them to make better decisions about whether they should be waiting or not.

2.2 Flow control issues

Clients uploading large message bodies to servers or via proxies which require authentication (especially multiple-round-trip schemas such as NTLM) have extreme difficulty knowing when to start sending message body content. The high number of transactions required to achieve a single transfer places a high burden on user time, and bandwidth.

The existing status code (100) intended to solve this problem does not work in the case where there are proxy agents in the request chain which also may require multiple-round-trip authentication.

2.3 Aim of this document

This document is intended to solve these problems by providing 1: a defined mechanism whereby intermediate or server agents which must engage in lengthy processing of client requests can provide visible progress notifications back to the user of the client software, and thereby avoiding user disconnect and re-request;

and 2: a more clear process for user agents, servers and proxies to use when dealing with flow control of client message bodies.
3. Status Code Definitions

3.1 100 Continue

The 100 Continue status code is currently defined in RFC2616.

The 100 Continue response message is generated by some HTTP server software to indicate to downstream agents to continue with a request. The intention of this message is to avoid the need to transfer a request body multiple times. However this approach does not work when there are chained agents which also require authentication.

As an example, a user agent submitting a POST request with a large message body (e.g. file upload to a webmail system) to a server which uses NTLM authentication can result in the message body being sent 3 times. With an intermediary in the chain also requiring NTLM authentication such as is common with many proxy implementations, this is increased to 5 times. Uploading a large file over a slow link 5 times simply to allow negotiation of credentials has obvious negative implications.

The 100 Continue response was intended to allow servers to make a 401 Authorization required response before the client started sending the message body. However to achieve this the user agent must wait to receive this response before sending the message body. There is a lack of uniformity in how this has been implemented.

Delays in upstream connection mean that frequently clients time out and send the body before the proxy has contacted the upstream server. If the server presents a challenge, the body must be received and dumped by the proxy. Cople this with the requirement of some auth schemes to have persistent connections, and the problem is exascerbated.

3.1.1 Further discussion of the flow-control problem

It is worthwhile to discuss why the normal 100 Continue status doesn’t work in all cases.

The reader is referred to [RFC4559] for more information on the NTLM authentication scheme for HTTP, but suffice it to say, it is a token exchange protool requiring a token to be presented by the client to the server, another token sent back, and a final token sent through.

Couple this with the initial request that elicited the original challenge, and you have on average 3 requests made to establish authentication.

In the case where there is a proxy that requires NTLM, and a server requiring NTLM, the sequence is as follows
NTLM requires a persistent connection. If any connection is broken then the session-based credentials expire and the process must start from scratch. There is no mechanism in HTTP to cleanly abort the transfer of a message body, and messages must be clearly delineated in order to allow for connection persistence.

Therefore the client must be PREVENTED from sending the message body until credentials are fully established all the way to the server.

This is why the 100 Continue status code is not suitable - it can only tell a client to proceed with sending the body, it can’t tell it to wait. Typical implementations assume that a server may never send a 100 Continue, so the client waits a short while before giving up. Delays in establishing the connection between the proxy and server mean that often the client will start to send the message body.

### 3.1.2 Use of 100 continue for progress messages

Whilst the 100 Continue response won’t solve the flow control problem it was intended for, it does ironically provide a handy means of providing a solution to our second problem.

**RFC2616** states that multiple 100 Continue responses may be sent by a server before the final response is sent. This presents an ideal opportunity to provide visible progress indication to a user.

This document makes no proposal to change the semantics or usage of this status response, except to add a tag to provide information to end users.
3.2 102 Defer Transmission

This document proposes the addition of a new status code. The concept of Defer is the opposite of the concept of continue. This is what enables the solution to the problem outlined in 3.1.

On receipt of a request message (PUT OR POST) indicating that a message body will be transmitted by the client, a server or proxy MAY respond immediately with the 102 Defer transmission status code IF the proxy or server knows that authentication will be required. This allows the proxy or server to prevent the user agent from sending the body until after all credentials have been established along the chain.

A client having a POST or PUT message to send with a message body MUST NOT send the message body immediately. The client MUST wait until either it has received a message from the server or proxy indicating it may proceed (100 Continue), or until a timer expires. If the client receives a 102 Defer Transmission response, it MUST no longer use a timer to determine when to send but MUST wait until a subsequent 100 Continue message is received.

A client or proxy MUST be prepared to receive multiple 102 Defer Transmission messages. Subsequent messages after the first one MUST be discarded.
4 Header field definitions

This document defines a new response header field to indicate progress. This field allows optional textual state indication as well as requiring numeric indication of completeness.

4.1 Progress

The Progress field MAY appear in any request message sent by a client or in any server or proxy-generated 1xx series of response messages. The Progress field MUST not appear in any other series of response messages.

A server that complies with this document upon receiving a request containing a Progress field MUST provide a response message (either 100 Continue or normal response appropriate for the request) within the time period specified in the client request Progress field data AND provide further periodic updates each elapsed interval until the final response to the request is sent.

Progress indication is worthless if it is not timely.

A proxy that complies with this document upon receiving a request containing a Progress field MUST forward the tag unchanged to any upstream agent. The proxy however is responsible for providing progress messages to the client, and in the event that no upstream notifications are available, it MUST generate progress messages whilst it is still prepared to process the request.

The proxy SHOULD pass any 100 Continue messages back through to the client unchanged. The proxy SHOULD ensure that no more than one such message is sent downstream in any requested interval to avoid avalanche of such messages in a multi-stage request chain. In general the upstream notifications are most interesting to user-agents, so these SHOULD take preference to any message generated by the intermediary.

The form of the field is defined below

4.1.1 Client request

Progress = "Progress" ":" SP interval SP lang

interval = the time in seconds in which the client expects a response

lang = language tag to indicate the preferred language of any text progress indications.

Example: Progress: 10 en-nz
4.1.2 1xx Responses

Progress = "Progress" ":" SP numeric [";" SP textual]

textual = any text

numeric = (percent | bytes-processed "/" bytes-total)

percent = "%" number

bytes-processed = number

bytes-total = number | "UNKNOWN"

The textual information is intended to be displayed verbatim to a user in an area usually associated with progress indication. The numeric-progress field can be used to display a progress bar and/or show how much data has been received by an upstream proxy.

Examples of its use are:

Progress: 20480/UNKNOWN; Generating content

This could be sent by a server to indicate progress of generation of content.

Progress: 20480/UNKNOWN; Download in progress

Could be sent by an upstream proxy that is retrieving a message body, has received 20480 bytes, and doesn’t know the content length.

Progress: 1200000/1200000; Download complete, scanning

Could be sent by an upstream proxy that has just completed retrieving a message body, has received 1200000 of 1200000 bytes, and is about to scan the content before sending it on to the client.

Progress: %25; Scanning content for viruses

Could be sent by an upstream proxy that is currently 25% through virus scanning of a message body.

Progress: %45; Enumerating mail folders

could be sent by a webmail server opening a user mailbox
5. Examples

5.1. Use of 102 Defer and 100 Continue for flow control

Case 1, CLIENT, PROXY and SERVER all support this spec, and PROXY and SERVER both require NTLM authentication

<table>
<thead>
<tr>
<th>CLIENT</th>
<th>PROXY</th>
<th>SERVER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 POST URI /HTTP/1,1  -&gt;</td>
<td>Content-Length: 51864356</td>
<td>Proxy-Connection: Keep-alive</td>
</tr>
<tr>
<td>2</td>
<td>&lt;- HTTP/1.1 102 Defer Transmission</td>
<td>Proxy-Connection: Keepalive</td>
</tr>
<tr>
<td>3</td>
<td>&lt;- HTTP/1.1 407 Authorization required</td>
<td>Proxy-Authentication: NTLM, Proxy-Connection: Keep-alive</td>
</tr>
<tr>
<td>4 POST URI /HTTP/1,1  -&gt;</td>
<td>Content-Length: 51864356</td>
<td>Proxy-Authentication: NTLM &lt;token&gt;, Proxy-Connection: Keep-alive</td>
</tr>
<tr>
<td>5</td>
<td>&lt;- HTTP/1.1 407 Authorization required</td>
<td>Proxy-Authentication: NTLM &lt;token&gt;, Proxy-Connection: Keep-alive</td>
</tr>
<tr>
<td>6 POST URI /HTTP/1,1  -&gt;</td>
<td>Content-Length: 51864356</td>
<td>Proxy-Authentication: NTLM &lt;token&gt;, Proxy-Connection: Keep-alive</td>
</tr>
<tr>
<td>7</td>
<td>POST URI /HTTP/1,1  -&gt;</td>
<td>Content-Length: 51864356</td>
</tr>
<tr>
<td>8</td>
<td>&lt;- HTTP/1.1 102 Defer Transmission</td>
<td>Connection: Keepalive</td>
</tr>
<tr>
<td>9</td>
<td>&lt;- HTTP/1.1 102 Defer Transmission</td>
<td>Proxy-Connection: Keep-alive</td>
</tr>
<tr>
<td>10</td>
<td>&lt;- HTTP/1.1 401 Authorization</td>
<td>Authorization: NTLM, Connection: Keep-alive</td>
</tr>
</tbody>
</table>
11  <- HTTP/1.1 401 Authorization
    Authorization: NTLM
    Proxy-Connection: Keep-alive

12 POST URI /HTTP/1,1   ->
    Content-Length: 51864356
    WWW-Authenticate: NTLM <token>
    Proxy-Connection: Keep-alive

13 POST URI /HTTP/1,1   ->
    Content-Length: 51864356
    WWW-Authenticate: NTLM <token>
    Connection: Keep-alive

14  <- HTTP/1.1 401 Authorization
    Authenticate: NTLM <token>
    Connection: Keep-alive

15  <- HTTP/1.1 401 Authorization
    Authorization: NTLM <token>
    Proxy-Connection: Keep-alive

16 POST URI /HTTP/1,1   ->
    Content-Length: 51864356
    WWW-Authenticate: NTLM <token>
    Proxy-Connection: Keep-alive

17 POST URI /HTTP/1,1   ->
    Content-Length: 51864356
    WWW-Authenticate: NTLM <token>
    Connection: Keep-alive

18  <- HTTP/1.1 100 Continue
    Connection: Keepalive

19  <- HTTP/1.1 100 Continue
    Proxy-Connection: Keep-alive

20 Client sends body->

21 Proxy sends body->

22  <- HTTP/1.1 200 OK

23  <- HTTP/1.1 200 OK

24 Done.

There are several scenarios of compatibility issue.

The client may or may not support this specification
The server may or may not support this specification
The proxy may or may not support this specification

There are cases where various parties need to auth or not:

* Proxy may need auth or not
* Server may need auth or not.

And finally, the connection may be directly between client and server

This makes a lot of combinations to discuss. However, if we start with simple 2-party communications, we can extend it, since a proxy is both a client and server

6.1 Client does not support this specification and server does.

in this case, there is no change to current widespread operation

6.1 Client supports this specification and server does not.

in this case, the client behaves as per RFC2616 and waits for a timeout value before sending the message body.
7. Implementation Notes

Most user agents provide screen real-estate to display progress, often in the "status bar" of the window. It is envisaged that the progress notifications outlined in this document would be shown in there.
8. Security Considerations

No security issues identified with use of this proposal.
9. Notes & TODO

* flesh out compatibility issues.

* further consider any possible security issues (none raised at this stage)
10. IANA Considerations

None.
11. References

11.1. Normative References


Author’s Address

Adrien de Croy
Qbik New Zealand Limited
PO Box 3548
Shortland St
Auckland
New Zealand

Email: adrien@qbik.com
URI: http://www.wingate.com/