INTERNET-DRAFT Mandatory
draft-frystyk-http-mandatory
H. Frystyk Nielsen, W3C
P. Leach, Microsoft
Scott Lawrence, Agranat Systems
Expires: July 20, 1998
Tuesday, January 20, 1998

Mandatory Extensions in HTTP

Status of this Document

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).

Distribution of this document is unlimited. Please send comments to the <ietf-http-ext@w3.org> mailing list. This list is archived at "http://lists.w3.org/Archives/Public/ietf-http-ext/".

The contribution of World Wide Web Consortium (W3C) staff is part of the W3C HTTP Activity (see "http://www.w3.org/Protocols/Activity").

Abstract

HTTP is used increasingly in applications that need more facilities than the standard version of the protocol provides, ranging from distributed authoring, collaboration, and printing, to various remote procedure call mechanisms. This document proposes the use of a mandatory extension mechanism designed to address the tension between private agreement and public specification and to accommodate extension of applications such as HTTP clients, servers, and proxies. The proposal associates each extension with a URI[2], and use a few new RFC 822[1] style header fields to carry the extension identifier and related information between the parties involved in an extended transaction.

Table of Contents

1. Introduction

The mandatory proposal is designed to accommodate dynamic extension of HTTP clients and servers by software components; and to address the tension between private agreement and public specification. The proposal uses features in HTTP/1.1 but is compatible with both HTTP/1.0 and HTTP/1.1 applications. The kind of extensions capable of being introduced range from:

- extending a single HTTP message;
- introducing new encodings;
- initiating HTTP-derived protocols for new applications; to...
- switching to protocols which, once initiated, run independent of the original protocol stack.

The proposal is intended to be used as follows:

- Some party designs and specifies an extension; the party assigns the extension an identifier, which is a URI, and makes one or more representations of the extension available at that address (see section 7).
- A party using an agent implementing the extension wishes to use it; the agent declares the use of the extension by referencing its URI in an extension declaration (see section 3).

2. Notational Conventions

This specification uses the same notational conventions and basic parsing constructs as RFC 2068[7]. In particular the BNF constructs "token", "quoted-string", "field-name", and "URI" in this document are to be interpreted as described in RFC 2068[7].

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this

document are to be interpreted as described in RFC 2119[8].

This proposal does not rely on particular features defined in URLs [3] that cannot potentially be expressed using URNs (see section 7). Therefore, the more generic term URI[2] is used throughout the specification.

3. Extension Declarations

An extension declaration can be used to indicate that an extension has been applied to a message and possibly to reserve a part of the header namespace identified by a header field prefix (see 3.1). The grammar for an extension declaration is as follows:

```
ext-decl        = <"> URI <"> [ ext-params ]
ext-params      = ";" namespace *( ext-extension )

namespace       = ";" "ns" "=" prefix
prefix          = 1*DIGIT "-"
ext-extension   = ";" token [ "=" ( token | quoted-string ) ]
```

An extension is identified by a URI. Extension identifier URIs can be either relative or absolute. Relative extension identifiers are interpreted relative to the IANA registry (see RFC 1808[4]). Examples of URIs are

```
"http://www.temporary.com/extension"
"rfc6534"
"Content-FooBar"
```

An extension declaration can be extended through the use of one or more ext-extension parameters. Unrecognized ext-extension parameters SHOULD be ignored and MUST NOT be removed by proxies when forwarding the extension declaration.

Note: In layered implementations, unknown ext-extension parameters should be passed to the upper layers as they may have other mechanisms of knowing the semantics of the parameters.

3.1 Header Field Prefixes

The header-prefix can be used to indicate that all header fields in the message matching the header-prefix value using string prefix-matching are introduced by this extension instance. This allows an extension instance to dynamically reserve a subspace of the header space in a protocol message in order to prevent header field name clashes. Agents SHOULD NOT reuse header-prefix values in the same message.

Examples of header-prefix values are

```
    1234-
    546-
    234345653-
```

Linear white space (LWS) MUST NOT be used between the 1*DIGIT and the "-". The format of the prefix using a combination of digits and the dash "-" guarantees that no extension declaration can reserve the whole header field name space.

Note: Old applications may introduce header fields independent of this extension mechanism, potentially conflicting with header fields introduced by the prefix mechanism. In order to minimize this risk, prefixes should contain at least 3 digits.

4. Extension Header Fields

This proposal introduces two types of extension declarations: mandatory and optional declarations. A mandatory extension declaration indicates that the ultimate recipient MUST consult and adhere to the rules given by the extension when processing the message or report an error (see section 5 and 6).

An optional extension declaration indicates that the ultimate recipient of the extension MAY consult and adhere to the rules given by the extension when processing the message, or ignore the extension declaration completely. An agent may not be able to distinguish whether the ultimate recipient does not understand an extension referred to by an optional extension or simply ignores the extension declaration.

There are two scopes for extensions declarations: Hop-by-hop and end-to-end. The scopes are distinguished by separate header field names so that multiple extensions with different scopes can be applied to the same message.

4.1 End-to-End Extensions

End-to-end and hop-by-hop. End-to-end declarations MUST be transmitted to the ultimate recipient of the declaration. The Man and the Opt general header fields are end-to-end header fields and are defined as follows:

```
mandatory       = "Man" ":" 1#ext-decl
optional        = "Opt" ":" 1#ext-decl
```

For example

```
HTTP/1.1 200 OK
Content-Length: 421
Opt: "http://www.digest.org/Digest"; ns=54-
54-digest: "4525dct344v@fsdfsg"
...
```

Proxies MAY act as both the initiator and the ultimate recipient of end-to-end extension declarations. It is outside the scope of this

specification to define how an agreement is reached between a party representing the proxy and the party on which behalf it can act, but for example, the parties may be within the same trust domain.

If a proxy is the ultimate recipient of a mandatory end-to-end extension declaration then it MUST handle that extension declaration as described in section 5. The proxy SHOULD remove all parts of the extension declaration from the message before forwarding it.

4.2 Hop-by-Hop Extensions

Hop-by-hop extension declarations are meaningful only for a single transport-level connection. The C-Man and the C-Opt general header field are hop-by-hop header fields and MUST NOT be communicated by proxies over further connections. The two headers have the following grammar:

```
c-mandatory     = "C-Man" ":" 1#ext-decl
c-optional      = "C-Opt" ":" 1#ext-decl
```

For example

```
GET / HTTP/1.1
Host: some.host
C-Man: "http://www.digest.org/ProxyAuth"; ns=23-
23-Credentials: "g5gj262jdw@4df"
Connection: C-Man, 23-Credentials
```

In HTTP/1.1, the C-Man and the C-Opt header field MUST be protected by a Connection header. That is, the header fields are to be included as Connection header directives (see section [7], section 14.10).

An agent MUST NOT send the C-Man or the C-Opt header field to an HTTP/1.0 proxy as it does not obey the HTTP/1.1 rules for parsing the Connection header field (see [7], section 19.7.1).

5. Mandatory HTTP Requests

An HTTP request is called a mandatory request if it includes at least one mandatory extension declaration (using the Man or the C-Man header fields). The method name of a mandatory request MUST be prefixed by "M-". For example, a client might express the binding rights-management constraints in an HTTP PUT request as follows:

```
M-PUT /a-resource HTTP/1.1
Man: "http://www.copyright.org/rights-management"; ns=43-
43-copyright: http://www.copyright.org/COPYRIGHT.html-
43-contributions: http://www.copyright.org/PATCHES.html
```

```
    Host: www.w3.org
    Content-Length: 1203
    Content-Type: text/html
```

```
<!doctype html ...
```

An HTTP server MUST NOT return a 2xx status-code without obeying all mandatory extension declaration(s) in a mandatory request. A mandatory HTTP request invalidates cached entries as described in [7], section 13.10.

The ultimate recipient of a mandatory HTTP request with the "M-" prefix on the method name MUST process the request by performing the following actions in the order they occur:

1. Identify all mandatory extension declarations (both hop-by-hop and end-to-end); the server MAY ignore optional declarations without affecting the result of the transaction;
2. Evaluate and process the extensions identified in 1) or if the extension declarations do not match the policy for accessing the resource then respond with a 510 (Not Extended) status-code (see section 6);
3. Strip the "M-" prefix from the method name and process the reminder of the request according to the semantics of the existing HTTP/1.1 method name as defined in [7].

An "M-" aware proxy that does not act as the ultimate recipient of a mandatory extension declaration MUST NOT remove the declaration or the "M-" method name prefix when forwarding the message.

The "M-" prefix is reserved by this proposal and MUST NOT be used by other HTTP extensions.

Note: Applications that do not understand the "M-" method name prefix should return 501 (Not Implemented) or turn themselves into a tunnel ([7]) in which case they do not take any part in the communication.

6. 510 Not Extended

The policy for accessing the resource has not been met in the request. The server SHOULD send back all the information necessary for the client to issue an extended request. It is outside the scope of this specification to specify how the extensions inform the client.

If the initial request already included the extensions requested in the 510 response, then the response indicates that access has been refused for those extension declarations.

If the 510 response contains the same set of extension policies as the prior response, then the client MAY present any entity included in the response to the user, since that entity may include relevant diagnostic information.

7. Publishing an Extension

While the protocol extension definition should be published at the address of the extension identifier, this is not a requirement of this specification. The only absolute requirement is that distinct names be used for distinct semantics. For example, one way to achieve this is to use a mid, cid, or uuid URI. The association between the extension identifier and the specification might be made by distributing a specification, which references the extension identifier.

It is strongly recommended that the integrity and persistence of the extension identifier is maintained and kept unquestioned throughout the lifetime of the extension. Care should be taken not to distribute conflicting specifications that reference the same name. Even when a URI is used to publish extension specifications, care must be taken that the specification made available at that address does not change significantly over time. One agent may associate the identifier with the old semantics, and another might associate it with the new semantics.

The extension definition may be made available in different representations ranging from

- a human-readable specification defining the extension semantics,
- downloadable code which implements the semantics defined by the extension,
- a formal interface description provided by the extension, to
- a machine-readable specification defining the extension semantics.

For example, a software component that implements the specification may reside at the same address as a human-readable specification (distinguished by content negotiation). The human-readable representation serves to document the extension and encourage deployment, while the software component allows clients and servers to be dynamically extended.

8. Security Considerations

- Dynamic installation of extension facilities as described in the introduction involves software written by one party (the provider of the implementation) to be executed under the authority of another (the party operating the host software). This opens the host party to a variety of "Trojan horse" attacks by the provider, or a malicious third party that forges implementations under a provider's name. See, for example RFC2046[6], section 4.5.2 for a discussion of these risks.

9. References

[1] D. H. Crocker. "Standard for the Format of ARPA Internet Text Messages", STD 11, RFC 822, UDEL, August 1982

[2] T. Berners-Lee, "Universal Resource Identifiers in WWW. A Unifying Syntax for the Expression of Names and Addresses of Objects on the Network as used in the World-Wide Web", RFC 1630, CERN, June 1994.

[3] T. Berners-Lee, L. Masinter, M. McCahill. "Uniform Resource Locators (URL)" RFC 1738, CERN, Xerox PARC, University of Minnesota, December 1994.

[4] R. Fielding, "Relative Uniform Resource Locators", RFC 1808, UC Irvine, June 1995.

[5] T. Berners-Lee, R. Fielding, H. Frystyk, "Hypertext Transfer Protocol -- HTTP/1.0", RFC 1945, W3C/MIT, UC Irvine, W3C/MIT, May 1996.

[6] N. Freed, N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, Innosoft, First Virtual, November 1996.

[7] R. Fielding, J. Gettys, J. C. Mogul, H. Frystyk, T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2068, U.C. Irvine, DEC W3C/MIT, DEC, W3C/MIT, W3C/MIT, January 1997

[8] S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, Harvard University, March 1997

[9] Y. Goland et al, "Extensions for Distributed Authoring and Versioning", Internet Draft, draft-jensen-webdav-ext-01, 26 March 1997. This is work in progress.

[10] H. F. Nielsen, D. Connolly, R. Khare, "PEP - an extension mechanism for HTTP", draft-http-pep-05.txt, November 21, 1997

10. Acknowledgements

Rohit Khare deserves special recognition for his efforts in commenting in the design phase of the protocol. Also thanks to Josh Cohen, Jim Gettys and all the people who have been involved in PEP.

11. Authors Addresses

Henrik Frystyk Nielsen
Technical Staff, World Wide Web Consortium
MIT Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139, USA
Email: frystyk@w3.org

Paul J. Leach
Microsoft Corporation
1 Microsoft Way

Redmond, WA 98052, USA
Email: paulle@microsoft.com

Scott Lawrence
Agranat Systems, Inc.
1345 **Main Street**
Waltham, MA 02154, USA
Email: lawrence@agranat.com

Appendices

12. Summary of Protocol Interactions

The following tables summarize the outcome of strength and scope rules of the mandatory proposal of compliant and non-compliant HTTP proxies and origin servers. The summary is intended as a guide and index to the text, but is necessarily cryptic and incomplete. This summary should never be used or referenced separately from the complete specification.

Table 1: Origin Server

Scope	Hop-by-hop		End-to-end	
Strength	Optional (may)	Required (must)	Optional (may)	Required (must)
Mandatory unsupported	Standard processing	501 (Not Implemented)	Standard processing	501 (Not Implemented)
Extension unsupported	Standard processing	510 (Not Extended)	Standard processing	510 (Not Extended)
Extension supported	Extended processing	Extended processing	Extended processing	Extended processing

Table 2: Proxy Server

Scope	Hop-by-hop		End-to-end	
Strength	Optional (may)	Required (must)	Optional (may)	Required (must)
Mandatory unsupported	Strip extension	501 (Not Implemented)	Forward extension	501 (Not Implemented) or tunnel
Extension unsupported	Strip extension	510 (Not Extended)	Forward extension	Forward extension

Extension supported	Extended processing and strip	Extended processing and strip	Extended processing, may strip	Extended processing, may strip

13. Examples

The following examples show various scenarios using mandatory in HTTP/1.1 requests and responses. Information not essential for illustrating the examples is left out (referred to as " ")

13.1 Client Queries Server for DAV

In this example, the purpose is to determine whether a server understands and supports the Distributed Authoring and Versioning (DAV) protocol extension [9]. By making the request mandatory (see section 5), the client forces the server to process the extension declaration and obey the extension or report an error.

```
M-GET /some.url HTTP/1.1
Host: some.host
Man: "http://www.dav.org"
...

HTTP/1.1 200 OK
...
```

The response shows that the server does understand. It is not possible to distinguish between querying about or using an extension - the extension declaration is identical. Whether it in fact is a query may depend on the request method name and request modifiers.

13.2 Server Uses ZipFlate Compression Extension

This example shows a server using the zipflate compression extension in a response:

```
GET /Index HTTP/1.1
Host: some.host

HTTP/1.1 200 OK
Man: "http://www.encoding.com/zipflate"
Cache-Control: no-transform
Vary: *
...
```

The response shows that the server uses the extension the response. The response includes the no-transform cache-control directive in order to avoid that proxies add their own content-coding to the message and a Vary header field indicating that a cache may not use the response to reply to a subsequent request without revalidation.