H. Frystyk Nielsen, W3C
draft-frystyk-http-extensions-01 P. Leach, Microsoft
Scott Lawrence, Agranat Systems

Expires: May 11, 1999 Wednesday, November 11, 1998

HTTP Extension Framework

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/".

Abstract

A wide range of applications have proposed various extensions of the HTTP protocol. Current efforts span an enormous range, including distributed authoring, collaboration, printing, and remote procedure call mechanisms. However, HTTP extensions are uncoordinated, and there has been no standard framework for defining extensions or keeping them separate from each other. This document describes a generic extension mechanism for HTTP/1.1, which is designed to address the tension between private agreement and public specification and to accommodate extension of applications using HTTP clients, servers, and proxies. The proposal associates each extension with a globally unique identifier, and uses HTTP header fields to carry the extension identifier and related information between the parties involved in the extended communication.

Table of Contents

1. Introduction

This proposal is designed to address the tension between private agreement and public specification; and to accommodate dynamic extension of HTTP clients and servers by software components. The kind of extensions capable of being introduced range from:

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

The proposal is intended to be used as follows:

- o Some party designs and specifies an extension; the party assigns the extension a globally unique address (URI), and makes one or more representations of the extension available at that address (see section 8).
- o An HTTP client or server that implements this extension mechanism (hereafter called an *agent*) declares the use of the extension by referencing its URI in an extension declaration in an HTTP message (see section 3).
- o The HTTP application which the extension declaration is intended for (hereafter called the *ultimate recipient*) can deduce how to properly interpret the extended message based on the extension declaration.

The proposal uses features in HTTP/1.1 but is compatible with HTTP/1.0 applications in such a way that extended applications can coexist with existing HTTP applications. Applications implementing this proposal MUST be based on HTTP/1.1 (or later versions of HTTP).

2. Notational Conventions

This specification uses the same notational conventions and basic parsing constructs as RFC 2068 [5]. In particular the BNF constructs "`token`", "`quoted-string`", "`Request-Line`", "`field-name`", and "`absoluteURI`" in this document are to be interpreted as described in RFC 2068 [5].

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 [6].

This proposal does not rely on particular features defined in URLs [8] that cannot potentially be expressed using URNs (see section 8). Therefore, the more generic term URI [8] 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).

This specification does not define any ramifications of applying an extension to a message nor whether two extensions can or cannot logically coexist within the same message. It is simply a framework for describing which extensions have been applied and what the ultimate recipient either must or may do in order to properly interpret any extension declarations within that message.

The grammar for an extension declaration is as follows:

```
ext-decl        = <"> ( absoluteURI | field-name ) <">
                  ";" namespace [ decl-extensions ]
decl-extensions = *( decl-ext )

namespace       = "ns" "=" header-prefix
header-prefix   = 2*DIGIT "-"
decl-ext        = ";" token [ "=" ( token | quoted-string ) ]
```

An extension is identified by an absolute, globally unique URI or a `field-name`. A `field-name` MUST specify a header field uniquely defined in an IETF Standards Track RFC [3]. A URI can unambiguously be distinguished from a field-name by the presence of a colon (":"). The support for header field names as extension identifiers provides a transition strategy from decentralized extensions to extensions defined by IETF Standards Track RFCs.

Examples of extension declarations are

```
"http://www.company.com/extension"; ns=11-
"Range"; ns=13-
```

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

3.1 Header Field Prefixes

The `header-prefix` are dynamically generated header field prefix strings that 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. Prefixes are primarily intended to avoid header field name conflicts and to allow multiple instances of a single extension using its own header fields to be applied to the same message without conflicting with each other.

Linear white space (LWS) MUST NOT be used between the digits and the dash ("-"). 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.

Agents MUST NOT reuse `header-prefix` values in the same message unless explicitly allowed by the extension (see section 4.1 for a discussion of the ultimate recipient of an extension declaration).

Clients SHOULD be as consistent as possible when generating `header-prefix` values as this facilitates use of the `Vary` header field in responses that vary as a function of the request extension declaration(s) (see [5], section 13.6).

Examples of `header-prefix` values are

```
12-
15-
23-
```

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 MUST contain at least 2 digits.

4. Extension Header Fields

This proposal introduces two types of extension declaration strength: mandatory and optional, and two types of extension declaration scope: hop-by-hop and end-to-end (see section 4.1 and 4.2).

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 reporting an error (see section 5 and 7).

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.

The combination of the declaration strength and scope defines a 2x2 matrix which is distinguished by four new general HTTP header fields: `Man`, `Opt`, `C-Man`, and `C-Opt.` (See section 4.1 and 4.2, and appendix 14 for a table of interactions with origin servers and proxies.)

The header fields are general header fields as they describe which extensions actually are applied to an HTTP message. Optional declarations MAY be applied to any HTTP message without any change to existing HTTP semantics. Mandatory declarations MUST be applied to a request message as described in section 5 and to a response message as described in section 6.

4.1 End-to-End Extensions

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=15-
15-digest: "snfksjgor2tsajkt52"
...
```

The ultimate recipient of a mandatory end-to-end extension declaration MUST handle that extension declaration as described in section 5 and 6.

4.2 Hop-by-Hop Extensions

Hop-by-hop extension declarations are meaningful only for a single HTTP connection. In HTTP/1.1, the `C-Man` and the `C-Opt` header field MUST be protected by a `Connection` header field. That is, the header fields are to be included as `Connection` header field directives (see [5], section 14.10). The two header fields 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=14-
14-Credentials="g5gj262jdw@4df"
Connection: C-Man, 14-Credentials
```

The ultimate recipient of a mandatory hop-by-hop extension declaration MUST handle that extension declaration as described in section 5 and 6.

4.3 Extension Response Header Fields

Two extension response header fields are used to indicate that a request containing mandatory extension declarations has been fulfilled by the ultimate recipient as described in section 5.1. The extension response header fields are exclusively intended to serve as extension acknowledgements and can not carry any other information.

The `Ext` header field is used to indicate that all end-to-end mandatory extension declarations in the request were fulfilled:

```
ext             = "Ext" ":"
```

The `C-Ext` response header field is used to indicate that all hop-by-hop mandatory extension declarations in the request were fulfilled.

```
c-ext           = "C-Ext" ":"
```

In HTTP/1.1, the `C-Ext` header field MUST be protected by a `Connection` header (see [5], section 14.10).

The `Ext` and the `C-Ext` header fields are not mutually exclusive, they can both occur within the same message as described in section 5.1.

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=16-
16-copyright: http://www.copyright.org/COPYRIGHT.html
16-contributions: http://www.copyright.org/PATCHES.html
Host: www.w3.org
Content-Length: 1203
Content-Type: text/html

<!doctype html ...
```

An ultimate recipient conforming to this specification receiving a mandatory request MUST process the request by performing the following actions in the order listed below:

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 processing the HTTP message;
2. Examine all extensions identified in 1) and determine if they are supported for this message. If not, respond with a 510 (Not Extended) status-code (see section 7);
3. If 2) did not result in a 510 (Not Extended) status code, then strip the "`M-`" prefix from the method name and process the remainder of the request according to the semantics of the extensions and of the existing HTTP/1.1 method name as defined in [5].
4. If the evaluation in 3) was successful and the mandatory request fulfilled, the server MUST respond as defined in section 5.1. A server MUST NOT fulfill a request without understanding and obeying all mandatory extension declaration(s) in a request.

A proxy that does not act as the ultimate recipient of a mandatory extension declaration MUST NOT remove the extension declaration or the "`M-`" method name prefix when forwarding the message. HTTP proxies that do not understand the "`M-`" method name prefix SHOULD return 501 (Not Implemented) or turn themselves into a tunnel (see [5]) in which case they do not take any part in the communication.

A server receiving an HTTP/1.0 (or earlier versions of HTTP) message that includes a `Connection` header MUST, for each connection-token in this field, remove and ignore any header field(s) from the message with the same name as the connection-token.

A server receiving a mandatory request including the "`M-`" method name prefix without any mandatory extension declarations to follow MUST return a 510 (Not Extended) response.

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

5.1 Fulfilling a Mandatory Request

A server MUST NOT claim to have fulfilled a mandatory request unless it understood and obeyed all the mandatory extension declarations in the request. This section defines a mechanism for conveying this information to the client in such a way that it interoperates with existing HTTP applications.

If any end-to-end mandatory extension declarations were among the fulfilled extensions then the server MUST include an `Ext` response header field in the response. In order to avoid that the `Ext` header field inadvertently is cached in an HTTP/1.1 cache, the response MUST contain a `no-cache` cache-control directive. If the response is otherwise cachable, the `no-cache` cache-control directive SHOULD be limited to only affect the `Ext` header field:

```
HTTP/1.1 200 OK
Ext:
Cache-Control: no-cache="Ext"
...
```

If the mandatory request has been forwarded by an HTTP/1.0 intermediary proxy then this is indicated either directly in the `Request-Line` or by the presence of an HTTP/1.1 `Via` header field. In this case, the server MUST include an `Expires` header field with a date equal to or earlier than the value of the `Date` header field. If the response is otherwise cachable by HTTP/1.1 caches, the server SHOULD include an appropriate `max-age` cache-control directive:

```
HTTP/1.1 200 OK
Date: Sun, 25 Oct 1998 08:12:31 GMT
Expires: Sun, 25 Oct 1998 08:12:31 GMT
Ext:
Cache-Control: no-cache="Ext", max-age=3600
...
```

If any hop-by-hop mandatory extension declarations were among the fulfilled extensions then the server MUST include a `C-Ext` response header field in the response. The `C-Ext` header field MUST be protected by a `Connection` header field (see [5], section 14.10).

```
HTTP/1.1 200 OK
C-Ext:
Connection: C-Ext
```

Note, that the `Ext` and `C-Ext` header fields are not mutually exclusive; they can be both be present in a response when fulfilling mandatory request containing both hop-by-hop as well as end-to-end mandatory extension declarations.

6. Mandatory HTTP Responses

A server MUST NOT include mandatory extension declarations in an HTTP response unless it is responding to a mandatory HTTP request whose definition allowed for the mandatory response or if the server has some apriori knowledge that the recipient can handle the extended response. A server MAY include optional extension declarations in any HTTP response (see section 4).

If a client is the ultimate recipient of a mandatory HTTP response containing mandatory extension declarations that either the client does not understand or does not want to use, then it SHOULD discard the complete response as if it were a 500 (Internal Server Error) response.

7. 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 510 response contains information about extensions that were not present in the initial request then the client MAY repeat the request if it has reason to believe it can fulfill the extension policy by modifying the request according to the information provided in the 510 response. Otherwise the client MAY present any entity included in the 510 response to the user, since that entity may include relevant diagnostic information.

8. 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 extension identifiers MUST be globally unique identifiers and that distinct names be used for distinct semantics.

Likewise, applications are not required to attempt resolving extension identifiers included in an extension declaration. The only absolute requirement is that an application MUST NOT claim conformance with an extension that it does not recognize regardless of whether it has tried to resolve the extension identifier or not. This document does not provide any policy for how long or how often an application may attempt to resolve an extension identifier.

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 be 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 an extension specification is made available at the address of the URI, care must be taken that the specification made available at that address does not change 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 (see for example [7]),
- 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.

9. Caching Considerations

Use of extensions using the syntax defined by this document may have additional implications on the cachability of HTTP response messages other than the ones described in section 5.1.

The originator of an extended message should be able to determine from the semantics of the extension whether or not the extension's presence impacts the cachability of the response message. If an extension does change the default cachability of the response, the originator MUST include a `Cache-Control` header field containing the cache directives corresponding to the desired result of the extended semantics.

10. 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 [4], section 4.5.2 for a discussion of these risks.

11. 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, R. Fielding, H. Frystyk, *"Hypertext Transfer Protocol -- HTTP/1.0"*, RFC 1945, W3C/MIT, UC Irvine, W3C/MIT, May 1996.

[3] S. Bradner, *"The Internet Standards Process -- Revision 3"*, RFC 2026, Harvard University, October 1996

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

[5] 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

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

[7] L. Masinter, *"Hyper Text Coffee Pot Control Protocol (HTCPCP/1.0)"*, RFC 2324, Xerox PARC, 1 April 1998

[8] T. Berners-Lee, R. Fielding, L. Masinter, *"Uniform Resource Identifiers (URI): Generic Syntax"*, RFC 2396, August 12, 1998

[9] H. F. Nielsen, D. Connolly, R. Khare, *"PEP - an extension mechanism for HTTP"*, draft-http-pep-05.txt, November 21, 1997. This work has expired

12. Acknowledgements

Roy Fielding, Rohit Khare, and Yaron Y. Goland deserve special recognition for their efforts in commenting in all phases of this specification. Also thanks to Josh Cohen, Ross Patterson, Jim Gettys, Larry Masinter, and to the people involved in PEP [9].

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

13. 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

14. 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) or tunnel	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

15. 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 "...")

15.1 User Agent to Origin Server

Table 3: User Agent directly to origin server

Client issues a request with one optional and one mandatory extension	M-GET <some-address> HTTP/1.1 Opt: <ext-1> Man: <ext-2> ...
Origin server accepts the mandatory extension but ignores the optional one. The client can not see in this case that the optional extension was ignored.	HTTP/1.1 200 OK Ext: Cache-Control: max-age=120, no-cache="Ext" ...

15.2 User Agent to Origin Server via HTTP/1.1 Proxy

These two examples show how an extended request interacts with an HTTP/1.1 proxy.

Table 4: HTTP/1.1 Proxy forwards extended request

Client issues a request with one optional and one mandatory hop-by-hop extension	M-GET <some-address> HTTP/1.1 C-Opt: <ext-1> C-Man: <ext-2> Connection: C-Opt, C-Man ...
HTTP/1.1 proxy forwards the request and takes out the connection headers	M-GET <some-address> HTTP/1.1 Via: 1.1 new ...
Origin server fails as the request does not contain any information belonging to the M-GET method	HTTP/1.1 510 Not Extended ...

Table 5: HTTP/1.1 Proxy does not forward extended request

Client issues a request with one optional and one mandatory hop-by-hop extension	M-GET <some-address> HTTP/1.1 C-Opt: <ext-1> C-Man: <ext-2> Connection: C-Opt, C-Man ...
HTTP/1.1 proxy refuses to forward the M-GET method and returns an error	HTTP/1.1 501 Not Implemented ...
Origin server never sees the extended request	

15.3 User Agent to Origin Server via HTTP/1.0 Proxy

These two examples show how an extended request interacts with an HTTP/1.0 proxy in the message path

Table 6: HTTP/1.0 Proxy forwards extended request

Client issues a request with one mandatory extension	M-GET <some-address> HTTP/1.1 Man: <ext-6> ...
HTTP/1.0 proxy forwards the request as a HTTP/1.0 request without changing the method	M-GET <some-address> HTTP/1.0 Man: <ext-6> ...
Origin server accepts <ext-6> and returns a 200 response and an extension acknowledgement. The response can be cached by HTTP/1.1 caches for 10 minutes.	HTTP/1.1 200 OK Ext: Date: Sun, 25 Oct 1998 08:12:31 GMT Expires: Sun, 25 Oct 1998 08:12:31 GMT Cache-Control: no-cache="Ext", max-age=600 ...

Table 7: HTTP/1.0 and HTTP/1.1 Proxy Chain

Client issues a request with one mandatory and one hop-by-hop optional extension	M-GET <some-address> HTTP/1.1 Man: <ext-1> C-Opt: <ext-2> Connection: C-Opt ...
HTTP/1.0 proxy forwards the request as HTTP/1.0 request without changing the method and without honoring the `Connection` header field.	M-GET <some-address> HTTP/1.0 Man: <ext-1> C-Opt: <ext-2> Connection: C-Man ...
HTTP/1.1 proxy deletes (and ignores) the optional extension and forwards the rest including a via header field. It also add it's own hop-by-hop mandatory extension	M-GET <some-address> HTTP/1.1 Man: <ext-1> C-Man: <ext-5> Connection: C-Man Via: 1.0 new ...
Origin server accepts both mandatory extensions. The response is not cachable by the HTTP/1.0 cache but can be cached for 1 hour by HTTP/1.1 caches.	HTTP/1.1 200 OK Ext: C-Ext Connection: C-Man Date: Sun, 25 Oct 1998 08:12:31 GMT Expires: Sun, 25 Oct 1998 08:12:31 GMT Cache-Control: no-cache="Ext", max-age=3600 ...
HTTP/1.1 proxy removes the hop-by-hop extension acknowledgement and forwards the remainder of the response.	HTTP/1.1 200 OK Ext: Date: Sun, 25 Oct 1998 08:12:31 GMT Expires: Sun, 25 Oct 1998 08:12:31 GMT Cache-Control: no-cache="Ext", max-age=3600 ...