H. Frystyk Nielsen, W3C
P. Leach, Microsoft
Scott Lawrence, Agranat Systems
draft-frystyk-http-extensions-00
Expires: February 07, 1999
Friday, August 07, 1998

HTTP Extension Framework for Mandatory and Optional Extensions

Status of this Document

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 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 an identifier, which is a URI, and makes one or more representations of the extension available at that address (see section 9).
- o An HTTP client, server, or proxy that implements the Mandatory 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 *ultimate recipient* of the extension declaration which can be the origin server, the user agent, or any intermediary in the request/response chain can based on the extension declaration deduce how to properly interpret the extended message.

The proposal uses features in HTTP/1.1 but is compatible with both HTTP/1.0 and HTTP/1.1 applications in such a way that extended applications can coexist with existing HTTP applications.

By providing a more robust framework for describing extensions, this proposal supersedes several existing extension mechanisms like the HTTP/1.1 Expect and Upgrade header fields as well as avoids existing problems with non-compliant CGI scripts handling unknown HTTP methods.

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[9].

This proposal does not rely on particular features defined in URLs [3] that cannot potentially be expressed using URNs (see section 9). 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).

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

namespace       = "ns" "=" header-prefix
header-prefix   = 2*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 MUST specify header-fields defined in an IETF RFC (see RFC 1808[4]). Examples of extension declarations are

```
"Content-FooBar"
"New-Registered-Header"
"http://www.temporary.com/extension"; ns=33-
```

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.

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.

Linear white space (LWS) MUST NOT be used between the digits 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.

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.

Agents SHOULD 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).

Examples of `header-prefix` values are

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

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 report an error (see section 5 and 8).

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=55-
55-digest: "snfksjgor2tsajkt52"
...
```

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"; Credentials="g5gj262jdw@4df"
Connection: C-Man
```

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]).

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 understanding and 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 are 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 the transaction;
2. If one or more mandatory extension declarations are present and the following is not true then respond with a 505 (HTTP Version Not Supported):
   - The request MUST NOT come from a HTTP/1.0 client; and
   - The request MUST NOT have any HTTP/1.0 clients indicated by the HTTP/1.1 `Via` header field.
3. If 2) is fulfilled then 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 8);

4. If the evaluation in 3) is successful (not resulting in a 510 (Not Extended) status code) then 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].
5. If one or more mandatory extension declarations were present in the original request and the evaluation in 3) was successful then the server MUST reply by sending a 102 (Extended) followed by a HTTP/1.1 response containing the appropriate HTTP header fields.

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.

An agent receiving an HTTP/1.0 (or lower-version) 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. Any "`M-`" method name prefix introduced as a result of discarded hop-by-hop extensions MUST be ignored and removed by a proxy 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 ([7]) in which case they do not take any part in the communication.

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

6. Mandatory HTTP Responses

A server SHOULD NOT include mandatory extension declarations in an HTTP response unless it is responding to a mandatory HTTP request. A server MAY include optional extension declarations in any HTTP response (see section 4).

If a client receives an HTTP response which contains a Mandatory extension declaration which it does not understand or does not want to use, it SHOULD treat it as if the message was of type "`application/octet-stream`".

7. 102 Extended

The server understands and is willing to comply with the client's extended request using mandatory extension declarations (section 4). The 102 (Extended) response is followed by a normal HTTP/1.1 style response indicating the final status code and parameters of the response.

The 102 (Extended) status code prevents that existing HTTP/1.1 servers using non-conformant CGI scripts mistakenly give the false impression that the extended request was fulfilled by responding with a 200 (Ok) response.

8. 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.

9. 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. For example, one way to achieve this is to use a *mid*, *cid*[8], or *uuid*[12] URI.

Likewise, applications are not required to attempt resolving extension identifiers included in extension declarations. 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 should 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 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.

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[6], 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, *"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] E. Levinson, "*Content-ID and Message-ID Uniform Resource Locators*", RFC 2111, March 1997

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

[10] Y. 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.

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

[12] Charlie Kindel, "*The uuid: URI scheme*", draft-kindel-uuid-uri-00.txt, November, 24 1997. This is work in progress

12. Acknowledgements

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

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 Client Requests Server to use an Extension

In this example, the client requires that the server supports and uses the extension identified by the URI " http://www.distributed.org/some.extension". 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.distributed.org/some.extension"
...

HTTP/1.1 102 Extended
```

```
HTTP/1.1 200 OK
...
```

The response shows that the server does understand the requested extension.

15.2 Server proposes the use of an Extension

By including an optional extension declaration in the response, the server indicates that the response has been extended but that it is OK if the client ignores the extension:

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

HTTP/1.1 200 OK
Opt: "http://www.cache.com/cache-index", ns=23-
23-index: "http://some.host/index"
...
```

The server has no direct mechanism of knowing whether the client accepted and used the optional extension declaration.