Transparent Content Negotiation in HTTP

Status of this Memo

Copyright Notice

ABSTRACT

   HTTP allows web site authors to put multiple versions of the same
   information under a single URL. Transparent content negotiation is
   an extensible negotiation mechanism, layered on top of HTTP, for
   automatically selecting the best version when the URL is accessed.
   This enables the smooth deployment of new web data formats and markup
   tags.

TABLE OF CONTENTS

1 Introduction

   HTTP allows web site authors to put multiple versions of the same
   information under a single URI. Each of these versions is called a
   'variant'. Transparent content negotiation is an extensible
   negotiation mechanism for automatically and efficiently retrieving
   the best variant when a GET or HEAD request is made. This enables
   the smooth deployment of new web data formats and markup tags.

   This specification defines transparent content negotiation as an
   extension on top of the HTTP/1.1 protocol [1]. However, use of this
   extension does not require use of HTTP/1.1: transparent content
   negotiation can also be done if some or all of the parties are
   HTTP/1.0 [2] systems.

   Transparent content negotiation is called 'transparent' because it
   makes all variants which exist inside the origin server visible to
   outside parties.

     Note: Some members of the IETF are currently undertaking a number
     of activities which are loosely related to this experimental
     protocol. First, there is an effort to define a protocol-
     independent registry for feature tags. The intention is that this
     experimental protocol will be one of the clients of the registry.
     Second, some research is being done on content negotiation systems
     for other transport protocols (like internet mail and internet fax)
     and on generalized negotiation systems for multiple transport
     protocols. At the time of writing, it is unclear if or when this
     research will lead to results in the form of complete negotiation
     system specifications. It is also unclear to which extent possible
     future specifications can or will re-use elements of this
     experimental protocol.

1.1 Background

   The addition of content negotiation to the web infrastructure has
   been considered important since the early days of the web. Among the
   expected benefits of a sufficiently powerful system for content
   negotiation are

     * smooth deployment of new data formats and markup tags will
       allow graceful evolution of the web

     * eliminating the need to choose between a 'state of the art
       multimedia homepage' and one which can be viewed by all web users

     * enabling good service to a wider range of browsing
       platforms (from low-end PDA's to high-end VR setups)

      * eliminating error-prone and cache-unfriendly
        User-Agent based negotiation

      * enabling construction of sites without 'click here for the X
        version' links

      * internationalization, and the ability to offer multi-lingual
        content without a bias towards one language.

2 Terminology

   The words "MUST", "MUST NOT", "SHOULD", "SHOULD NOT", and "MAY" in
   this document are to be interpreted as described in RFC 2119 [4].

   This specification uses the term 'header' as an abbreviation for for
   'header field in a request or response message'.

2.1 Terms from HTTP/1.1

   This specification mostly uses the terminology of the HTTP/1.1
   specification [1]. For the convenience of the reader, this section
   reproduces some key terminology definition from [1].

   request
     An HTTP request message.

   response
     An HTTP response message.

   resource
     A network data object or service that can be identified by a URI.
     Resources may be available in multiple representations (e.g.
     multiple languages, data formats, size, resolutions) or vary in
     other ways.

   content negotiation
     The mechanism for selecting the appropriate representation when
     servicing a request.

   client
     A program that establishes connections for the purpose of sending
     requests.

   user agent
     The client which initiates a request. These are often browsers,
     editors, spiders (web-traversing robots), or other end user tools.

   server
     An application program that accepts connections in order to service
     requests by sending back responses. Any given program may be
     capable of being both a client and a server; our use of these terms
     refers only to the role being performed by the program for a
     particular connection, rather than to the program's capabilities in
     general. Likewise, any server may act as an origin server, proxy,
     gateway, or tunnel, switching behavior based on the nature of each
     request.

   origin server
     The server on which a given resource resides or is to be created.

   proxy
     An intermediary program which acts as both a server and a client
     for the purpose of making requests on behalf of other clients.
     Requests are serviced internally or by passing them on, with
     possible translation, to other servers. A proxy must implement
     both the client and server requirements of this specification.

   age
     The age of a response is the time since it was sent by, or
     successfully validated with, the origin server.

   fresh
     A response is fresh if its age has not yet exceeded its freshness
     lifetime.

2.2 New terms

   transparently negotiable resource
     A resource, identified by a single URI, which has multiple
     representations (variants) associated with it. When servicing a
     request on its URI, it allows selection of the best representation
     using the transparent content negotiation mechanism. A
     transparently negotiable resource always has a variant list bound
     to it, which can be represented as an Alternates header (defined in
     section 8.3).

   variant list
     A list containing variant descriptions, which can be bound to a
     transparently negotiable resource.

variant description
  A machine-readable description of a variant resource, usually found
  in a variant list. A variant description contains the variant
  resource URI and various attributes which describe properties of
  the variant. Variant descriptions are defined in section 5.

variant resource
  A resource from which a variant of a negotiable resource can be
  retrieved with a normal HTTP/1.x GET request, i.e. a GET request
  which does not use transparent content negotiation.

neighboring variant
  A variant resource is called a neighboring variant resource of some
  transparently negotiable HTTP resource if the variant resource has
  a HTTP URL, and if the absolute URL of the variant resource up to
  its last slash equals the absolute URL of the negotiable resource
  up to its last slash, where equality is determined with the URI
  comparison rules in section 3.2.3 of [1]. The property of being a
  neighboring variant is important because of security considerations
  (section 14.2). Not all variants of a negotiable resource need to
  be neighboring variants. However, access to neighboring variants
  can be more highly optimized by the use of remote variant selection
  algorithms (section 7) and choice responses (section 10.2).

remote variant selection algorithm
  A standardized algorithm by which a server can sometimes choose a
  best variant on behalf of a negotiating user agent. The algorithm
  typically computes whether the Accept- headers in the request
  contain sufficient information to allow a choice, and if so, which
  variant is the best variant. The use of a remote algorithm can
  speed up the negotiation process.

list response
  A list response returns the variant list of the negotiable
  resource, but no variant data. It can be generated when the server
  does not want to, or is not allowed to, return a particular best
  variant for the request. List responses are defined in section
  10.1.

choice response
  A choice response returns a representation of the best variant for
  the request, and may also return the variant list of the negotiable
  resource. It can be generated when the server has sufficient
  information to be able to choose the best variant on behalf the
  user agent, but may only be generated if this best variant is a
  neighboring variant. Choice responses are defined in section 10.2.

adhoc response
  An adhoc response can be sent by an origin server as an extreme
  measure, to achieve compatibility with a non-negotiating or buggy
  client if this compatibility cannot be achieved by sending a list
  or choice response. There are very little requirements on the
  contents of an adhoc response. Adhoc responses are defined in
  section 10.3.

Accept- headers
  The request headers: Accept, Accept-Charset, Accept-Language, and
  Accept-Features.

supports transparent content negotiation
  From the viewpoint of an origin server or proxy, a user agent
  supports transparent content negotiation if and only if it sends a
  Negotiate header (section 8.4) which indicates such support.

server-side override
  If a request on a transparently negotiated resource is made by a
  client which supports transparent content negotiation, an origin
  server is said to perform a server-side override if the server
  ignores the directives in the Negotiate request header, and instead
  uses a custom algorithm to choose an appropriate response. A
  server-side override can sometimes be used to work around known
  client bugs. It could also be used by protocol extensions on top
  of transparent content negotiation.

3 Notation

  The version of BNF used in this document is taken from [1], and many
  of the nonterminals used are defined in [1]. Note that the
  underlying charset is US-ASCII.

  One new BNF construct is added:

      1%rule

  stands for one or more instances of "rule", separated by whitespace:

      1%rule = rule *(1*LWS rule)

  This specification also introduces

      number = 1*DIGIT

      short-float = 1*3DIGIT ["." 0*3DIGIT]

This specification uses the same conventions as in [1] (see section 1.2 of [1]) for defining the significance of each particular requirement.

4 Overview

This section gives an overview of transparent content negotiation. It starts with a more general discussion of negotiation as provided by HTTP.

4.1 Content negotiation

HTTP/1.1 allows web site authors to put multiple versions of the same information under a single resource URI. Each of these versions is called a 'variant'. For example, a resource http://x.org/paper could bind to three different variants of a paper:

        1. HTML, English
        2. HTML, French
        3. Postscript, English

Content negotiation is the process by which the best variant is selected if the resource is accessed. The selection is done by matching the properties of the available variants to the capabilities of the user agent and the preferences of the user.

It has always been possible under HTTP to have multiple representations available for one resource, and to return the most appropriate representation for each subsequent request. However, HTTP/1.1 is the first version of HTTP which has provisions for doing this in a cache-friendly way. These provisions include the Vary response header, entity tags, and the If-None-Match request header.

4.2 HTTP/1.0 style negotiation scheme

The HTTP/1.0 protocol elements allow for a negotiation scheme as follows:

    Server _____ proxy _____ proxy _____ user
    x.org   cache   cache   agent

      < ----------------------------------
      | GET http://x.org/paper
      | Accept- headers
    choose
      |
       -------------------------------- >
               Best variant
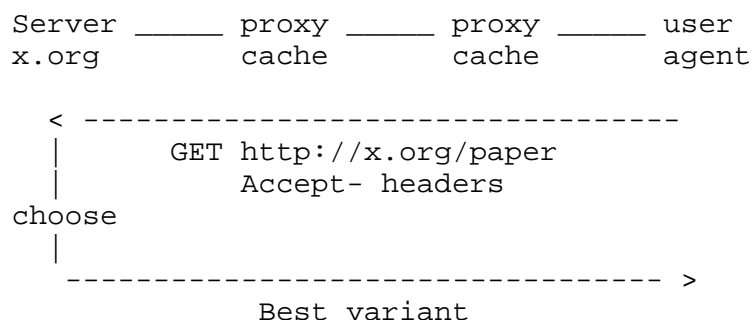
   When the resource is accessed, the user agent sends (along with its
   request) various Accept- headers which express the user agent
   capabilities and the user preferences. Then the origin server uses
   these Accept- headers to choose the best variant, which is returned
   in the response.

   The biggest problem with this scheme is that it does not scale well.
   For all but the most minimal user agents, Accept- headers expressing
   all capabilities and preferences would be very large, and sending
   them in every request would be hugely inefficient, in particular
   because only a small fraction of the resources on the web have
   multiple variants.

4.3 Transparent content negotiation scheme

   The transparent content negotiation scheme eliminates the need to
   send huge Accept- headers, and nevertheless allows for a selection
   process that always yields either the best variant, or an error
   message indicating that user agent is not capable of displaying any
   of the available variants.

   Under the transparent content negotiation scheme, the server sends a
   list with the available variants and their properties to the user
   agent. An example of a list with three variants is

      {"paper.1" 0.9 {type text/html} {language en}},
      {"paper.2" 0.7 {type text/html} {language fr}},
      {"paper.3" 1.0 {type application/postscript} {language en}}

   The syntax and semantics of the variant descriptions in this list are
   covered in section 5. When the list is received, the user agent can
   choose the best variant and retrieve it. Graphically, the
   communication can be represented as follows:

```
    Server _____  proxy _____  proxy _____  user
    x.org         cache        cache        agent

      < --------------------------------
      |        GET http://x.org/paper
      |
      |------------------------------- >          [list response]
               return of list           |
                                       choose
                                         |
      < --------------------------------
      |   GET http://x.org/paper.1
      |
       ------------------------------- >          [normal response]
             return of paper.1
```
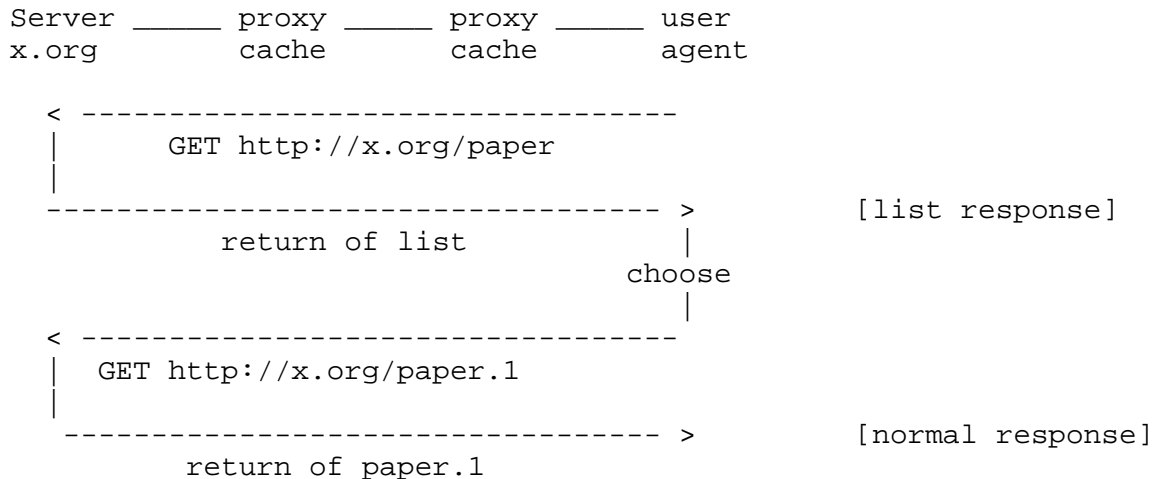
The first response returning the list of variants is called a 'list
response'. The second response is a normal HTTP response: it does
not contain special content negotiation related information. Only
the user agent needs to know that the second request actually
retrieves a variant. For the other parties in the communication, the
second transaction is indistinguishable from a normal HTTP
transaction.

With this scheme, information about capabilities and preferences is
only used by the user agent itself. Therefore, sending such
information in large Accept- headers is unnecessary. Accept- headers
do have a limited use in transparent content negotiation however; the
sending of small Accept- headers can often speed up the negotiation
process. This is covered in section 4.4.

List responses are covered in section 10.1. As an example, the list
response in the above picture could be:

```
   HTTP/1.1 300 Multiple Choices
   Date: Tue, 11 Jun 1996 20:02:21 GMT
   TCN: list
   Alternates: {"paper.1" 0.9 {type text/html} {language en}},
               {"paper.2" 0.7 {type text/html} {language fr}},
               {"paper.3" 1.0 {type application/postscript}
                   {language en}}
   Vary: negotiate, accept, accept-language
   ETag: "blah;1234"
   Cache-control: max-age=86400
   Content-Type: text/html
   Content-Length: 227
   <h2>Multiple Choices:</h2>
   <ul>
```

```
      <li><a href=paper.1>HTML, English version</a>
      <li><a href=paper.2>HTML, French version</a>
      <li><a href=paper.3>Postscript, English version</a>
      </ul>
```

   The Alternates header in the response contains the variant list. The
   Vary header is included to ensure correct caching by plain HTTP/1.1
   caches (see section 10.6). The ETag header allows the response to be
   revalidated by caches, the Cache-Control header controls this
   revalidation. The HTML entity included in the response allows the
   user to select the best variant by hand if desired.

4.4 Optimizing the negotiation process

   The basic transparent negotiation scheme involves two HTTP
   transactions: one to retrieve the list, and a second one to retrieve
   the chosen variant. There are however several ways to 'cut corners'
   in the data flow path of the basic scheme.

   First, caching proxies can cache both variant lists and variants.
   Such caching can reduce the communication overhead, as shown in the
   following example:

```
      Server _____ proxy _____ proxy _____ user
      x.org          cache         cache            agent

                                  < --------------
                                  |   GET ../paper
                                  |
                                  has the list
                                  in cache
                                    |
                                     ------------  >  [list response]
                                                list |
                                                     |
                                                  choose
                                                     |
                    < --------------------------
                    |    GET ../paper.1
                    |
                 has the variant
                 in cache
                    |
                     ------------------------- >  [normal response]
                          return of paper.1
```
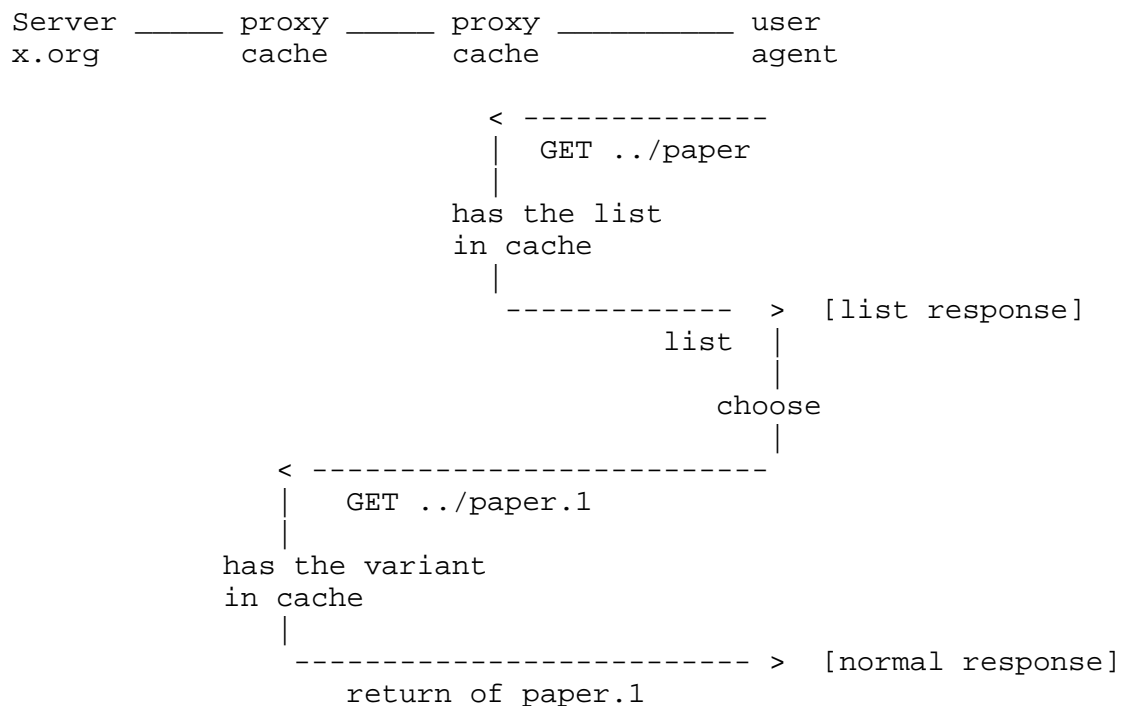
Second, the user agent can send small Accept- headers, which may
contain enough information to allow the server to choose the best
variant and return it directly.

```
   Server _____ proxy _____ proxy _____ user
   x.org         cache        cache        agent

    < --------------------------------
    |       GET http://x.org/paper
    |         small Accept- headers
    |
  able to choose on
  behalf of user agent
    |
     -------------------------------- >     [choice response]
          return of paper.1 and list
```
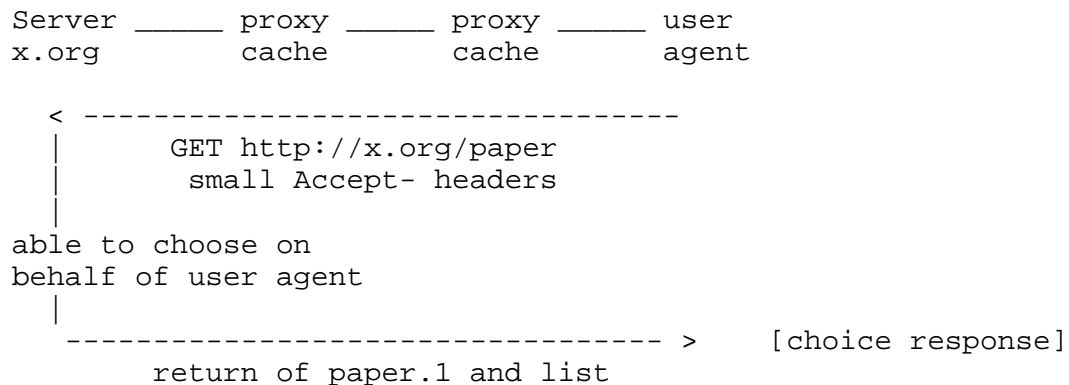
This choosing based on small Accept- headers is done with a 'remote
variant selection algorithm'. Such an algorithm takes the variant
list and the Accept- headers as input. It then computes whether the
Accept- headers contain sufficient information to choose on behalf of
the user agent, and if so, which variant is the best variant. If the
best variant is a neighboring variant, it may be returned, together
with the variant list, in a choice response.

A server may only choose on behalf of a user agent supporting
transparent content negotiation if the user agent explicitly allows
the use of a particular remote variant selection algorithm in the
Negotiate request header. User agents with sophisticated internal
variant selection algorithms may want to disallow a remote choice, or
may want to allow it only when retrieving inline images. If the
local algorithm of the user agent is superior in only some difficult
areas of negotiation, it is possible to enable the remote algorithm
for the easy areas only. More information about the use of a remote
variant selection algorithm can be found in [3].

Choice responses are covered in section 10.2. For example, the
choice response in the above picture could be:

```
  HTTP/1.1 200 OK
  Date: Tue, 11 Jun 1996 20:05:31 GMT
  TCN: choice
  Content-Type: text/html
  Last-Modified: Mon, 10 Jun 1996 10:01:14 GMT
  Content-Length: 5327
  Cache-control: max-age=604800
  Content-Location: paper.1
  Alternates: {"paper.1" 0.9 {type text/html} {language en}},
```

```
              {"paper.2" 0.7 {type text/html} {language fr}},
              {"paper.3" 1.0 {type application/postscript}
                  {language en}}
   Etag: "gonkyyyy;1234"
   Vary: negotiate, accept, accept-language
   Expires: Thu, 01 Jan 1980 00:00:00 GMT

   <title>A paper about ....
```

Finally, the above two kinds of optimization can be combined; a
caching proxy which has the list will sometimes be able to choose on
behalf of the user agent. This could lead to the following
communication pattern:

```
   Server _____ proxy _____ proxy _____ user
    x.org       cache       cache            agent


                        < ---------------
                        |   GET ../paper
                        |   small Accept
                        |
                    able to choose
                       on behalf
                        |
          < ----------
          |   GET ../paper.1
          |
           ---------- >   [normal response]
             paper.1  |
                       ---------------- >   [choice response]
                         paper.1 and list
```
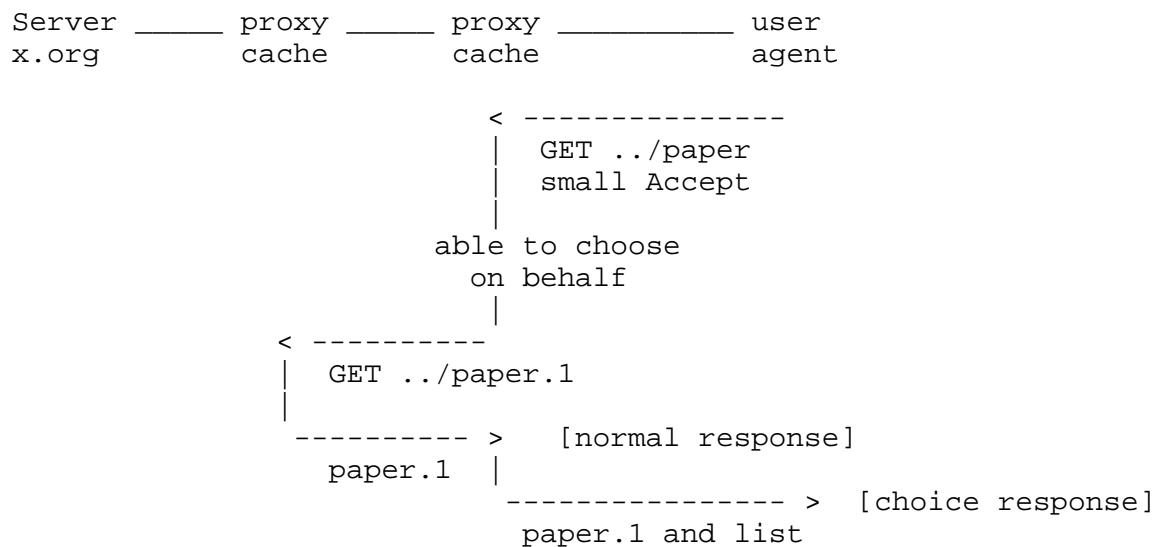
Note that this cutting of corners not only saves bandwidth, it also
eliminates delays due to packet round trip times, and reduces the
load on the origin server.

4.5 Downwards compatibility with non-negotiating user agents

To handle requests from user agents which do not support transparent
content negotiation, this specification allows the origin server to
revert to a HTTP/1.0 style negotiation scheme. The specification of
heuristics for such schemes is beyond the scope of this document.

4.6 Retrieving a variant by hand

   It is always possible for a user agent to retrieve the variant list
   which is bound to a negotiable resource. The user agent can use this
   list to make available a menu of all variants and their
   characteristics to the user. Such a menu allows the user to randomly
   browse other variants, and makes it possible to manually correct any
   sub-optimal choice made by the automatic negotiation process.

4.7 Dimensions of negotiation

   Transparent content negotiation defines four dimensions of
   negotiation:

      1. Media type (MIME type)
      2. Charset
      3. Language
      4. Features

   The first three dimensions have traditionally been present in HTTP.
   The fourth dimension is added by this specification. Additional
   dimensions, beyond the four mentioned above, could be added by future
   specifications.

   Negotiation on the content encoding of a response (gzipped,
   compressed, etc.) is left outside of the realm of transparent
   negotiation.   See section 10.8 for more information.

4.8 Feature negotiation

   Feature negotiation intends to provide for all areas of negotiation
   not covered by the type, charset, and language dimensions. Examples
   are negotiation on

      * HTML extensions
      * Extensions of other media types
      * Color capabilities of the user agent
      * Screen size
      * Output medium (screen, paper, ...)
      * Preference for speed vs. preference for graphical detail

   The feature negotiation framework (section 6) is the principal means
   by which transparent negotiation offers extensibility; a new
   dimension of negotiation (really a sub-dimension of the feature
   dimension) can be added without the need for a new standards effort
   by the simple registration of a 'feature tag'.

4.9 Length of variant lists

   As a general rule, variant lists should be short: it is expected that
   a typical transparently negotiable resource will have 2 to 10
   variants, depending on its purpose. Variant lists should be short
   for a number of reasons:

      1. The user must be able to pick a variant by hand to correct a
         bad automatic choice, and this is more difficult with a long
         variant list.

      2. A large number of variants will decrease the efficiency of
         internet proxy caches.

      3. Long variant lists will make some transparently negotiated
         responses longer.

   In general, it is not desirable to create a transparently negotiable
   resource with hundreds of variants in order to fine-tune the
   graphical presentation of a resource. Any graphical fine-tuning
   should be done, as much as possible, by using constructs which act at
   the user agent side, for example

      <center><img src=titlebanner.gif width=100%
      alt="MegaBozo Corp"></center>

   In order to promote user agent side fine tuning, which is more
   scalable than fine tuning over the network, user agents which
   implement a scripting language for content rendering are encouraged
   to make the availability of this language visible for transparent
   content negotiation, and to allow rendering scripts to access the
   capabilities and preferences data used for content negotiation, as
   far as privacy considerations permit this.

4.10 Relation with other negotiation schemes

   The HTTP/1.x protocol suite allows for many different negotiation
   mechanisms. Transparent content negotiation specializes in scalable,
   interoperable negotiation of content representations at the HTTP
   level. It is intended that transparent negotiation can co-exist with
   other negotiation schemes, both open and proprietary, which cover
   different application domains or work at different points in the
   author-to-user chain. Ultimately, it will be up to the resource
   author to decide which negotiation mechanism, or combination of
   negotiation mechanisms, is most appropriate for the task at hand.

5 Variant descriptions

5.1 Syntax

   A variant can be described in a machine-readable way with a variant
   description.

        variant-description =
                "{" <"> URI <"> source-quality *variant-attribute"}"

        source-quality = qvalue

        variant-attribute = "{" "type" media-type "}"
                          | "{" "charset" charset "}"
                          | "{" "language" 1#language-tag "}"
                          | "{" "length" 1*DIGIT "}"
                          | "{" "features" feature-list "}"
                          | "{" "description"
                                    quoted-string [language-tag] "}"
                          | extension-attribute

        extension-attribute = "{" extension-name extension-value "}"
        extension-name   = token
        extension-value   = *(token | quoted-string | LWS
                                | extension-specials)

        extension-specials =
                        <any element of tspecials except <"> and "}">

   The feature-list syntax is defined in section 6.4.

   Examples are

      {"paper.2" 0.7 {type text/html} {language fr}}

      {"paper.5" 0.9 {type text/html} {features tables}}

      {"paper.1" 0.001}

   The various attributes which can be present in a variant description
   are covered in the subsections below. Each attribute may appear only
   once in a variant description.

5.2 URI

   The URI attribute gives the URI of the resource from which the
   variant can be retrieved with a GET request. It can be absolute or
   relative to the Request-URI. The variant resource may vary (on the

Cookie request header, for example), but MUST NOT engage in
transparent content negotiation itself.

5.3 Source-quality

The source-quality attribute gives the quality of the variant, as a
representation of the negotiable resource, when this variant is
rendered with a perfect rendering engine on the best possible output
medium.

If the source-quality is less than 1, it often expresses a quality
degradation caused by a lossy conversion to a particular data format.
For example, a picture originally in JPEG form would have a lower
source quality when translated to the XBM format, and a much lower
source quality when translated to an ASCII-art variant. Note
however, that degradation is a function of the source; an original
piece of ASCII-art may degrade in quality if it is captured in JPEG
form.

The source-quality could also represent a level of quality caused by
skill of language translation, or ability of the used media type to
capture the intended artistic expression.

Servers should use the following table a guide when assigning source
quality values:

```
1.000   perfect representation
0.900   threshold of noticeable loss of quality
0.800   noticeable, but acceptable quality reduction
0.500   barely acceptable quality
0.300   severely degraded quality
0.000   completely degraded quality
```

The same table can be used by local variant selection algorithms (see
appendix 19) when assigning degradation factors for different content
rendering mechanisms. Note that most meaningful values in this table
are close to 1. This is due to the fact that quality factors are
generally combined by multiplying them, not by adding them.

When assigning source-quality values, servers should not account for
the size of the variant and its impact on transmission and rendering
delays; the size of the variant should be stated in the length
attribute and any size-dependent calculations should be done by the
variant selection algorithm. Any constant rendering delay for a
particular media type (for example due to the startup time of a
helper application) should be accounted for by the user agent, when
assigning a quality factor to that media type.

5.4 Type, charset, language, and length

   The type attribute of a variant description carries the same
   information as its Content-Type response header counterpart defined
   in [1], except for any charset information, which MUST be carried in
   the charset attribute. For, example, the header

      Content-Type: text/html; charset=ISO-8859-4

   has the counterpart attributes

      {type text/html} {charset ISO-8859-4}

   The language and length attributes carry the same information as
   their Content-* response header counterparts in [1]. The length
   attribute, if present, MUST thus reflect the length of the variant
   alone, and not the total size of the variant and any objects inlined
   or embedded by the variant.

   Though all of these attributes are optional, it is often desirable to
   include as many attributes as possible, as this will increase the
   quality of the negotiation process.

      Note: A server is not required to maintain a one-to-one
      correspondence between the attributes in the variant description
      and the Content-* headers in the variant response. For example,
      if the variant description contains a language attribute, the
      response does not necessarily have to contain a Content-Language
      header. If a Content-Language header is present, it does not have
      to contain an exact copy of the information in the language
      attribute.

5.5 Features

   The features attribute specifies how the presence or absence of
   particular feature tags in the user agent affects the overall quality
   of the variant. This attribute is covered in section 6.4.

5.6 Description

   The description attribute gives a textual description of the variant.
   It can be included if the URI and normal attributes of a variant are
   considered too opaque to allow interpretation by the user. If a user
   agent is showing a menu of available variants compiled from a variant
   list, and if a variant has a description attribute, the user agent
   SHOULD show the description attribute of the variant instead of
   showing the normal attributes of the variant. The description field
   uses the UTF-8 character encoding scheme [5], which is a superset of

US-ASCII, with ""%" HEX HEX" encoding. The optional language tag MAY
be used to specify the language used in the description text.

5.7 Extension-attribute

The extension-attribute allows future specifications to incrementally
define dimensions of negotiation which cannot be created by using the
feature negotiation framework, and eases content negotiation
experiments. In experimental situations, servers MUST ONLY generate
extension-attributes whose names start with "x-". User agents SHOULD
ignore all extension attributes they do not recognize. Proxies MUST
NOT run a remote variant selection algorithm if an unknown extension
attribute is present in the variant list.

6 Feature negotiation

This section defines the feature negotiation mechanism. Feature
negotiation has been introduced in section 4.8. Appendix 19 contains
examples of feature negotiation.

6.1 Feature tags

A feature tag (ftag) identifies something which can be negotiated on,
for example a property (feature) of a representation, a capability
(feature) of a user agent, or the preference of a user for a
particular type of representation. The use of feature tags need not
be limited to transparent content negotiation, and not every feature
tag needs to be usable in the HTTP transparent content negotiation
framework.

    ftag = token | quoted-string

    Note: A protocol-independent system for feature tag registration
    is currently being developed in the IETF. This specification does
    not define any feature tags. In experimental situations, the use
    of tags which start with "x." is encouraged.

Feature tags are used in feature sets (section 6.2) and in feature
predicates (section 6.3). Feature predicates are in turn used in
features attributes (section 6.4), which are used in variant
descriptions (section 5). Variant descriptions can be transmitted in
Alternates headers (section 8.3).

The US-ASCII charset is used for feature tags. Feature tag
comparison is case-insensitive. A token tag XYZ is equal to a
quoted-string tag "XYZ". Examples are

    tables, fonts, blebber, wolx, screenwidth, colordepth

An example of the use of feature tags in a variant description is:

    {"index.html" 1.0 {type text/html} {features tables frames}}

This specification follows general computing practice in that it
places no restrictions on what may be called a feature. At the
protocol level, this specification does not distinguish between
different uses of feature tags: a tag will be processed in the same
way, no matter whether it identifies a property, capability, or
preference. For some tags, it may be fluid whether the tag
represents a property, preference, or capability. For example, in
content negotiation on web pages, a "textonly" tag would identify a
capability of a text-only user agent, but the user of a graphical
user agent may use this tag to specify that text-only content is
preferred over graphical content.

6.1.1 Feature tag values

The definition of a feature tag may state that a feature tag can have
zero, one, or more values associated with it. These values
specialize the meaning of the tag. For example, a feature tag
'paper' could be associated with the values 'A4' and 'A5'.

    tag-value = token | quoted-string

The US-ASCII charset is used for feature tag values. Equality
comparison for tag values MUST be done with a case-sensitive, octet-
by-octet comparison, where any ""%" HEX HEX" encodings MUST be
processed as in [1]. A token value XYZ is equal to a quoted-string
value "XYZ".

6.2 Feature sets

The feature set of a user agent is a data structure which records the
capabilities of the user agent and the preferences of the user.

Feature sets are used by local variant selection algorithms (see
appendix 19 for an example). A user agent can use the Accept-
Features header (section 8.2) to make some of the contents of its
feature set known to remote variant selection algorithms.

Structurally, a feature set is a possibly empty set, containing
records of the form

    (feature tag , set of feature tag values)

If a record with a feature tag is present in the set, this means that
the user agent implements the corresponding capability, or that the
user has expressed the corresponding preference.

Each record in a feature set has a, possibly empty, set of tag
values. For feature tags which cannot have values associated with
it, this set is always empty. For feature tags which can have zero,
one, or more values associated with it, this set contains those
values currently associated with the tag. If the set of a feature
tag T has the value V in it, it is said that 'the tag T is present
with the value V'.

This specification does not define a standard notation for feature
sets. An example of a very small feature set, in a mathematical
notation, is

```
{ ( "frames" , { } ) ,
  ( "paper"  , { "A4" , "A5" } )
}
```

As feature registration is expected to be an ongoing process, it is
generally not possible for a user agent to know the meaning of all
feature tags it can possibly encounter in a variant description. A
user agent SHOULD treat all features tags unknown to it as absent
from its feature set.

A user agent may change the contents of its feature set depending on
the type of request, and may also update it to reflect changing
conditions, for example a change in the window size. Therefore, when
considering feature negotiation, one usually talks about 'the feature
set of the current request'.

6.3 Feature predicates

Feature predicates are predicates on the contents of feature sets.
They appear in the features attribute of a variant description.

```
fpred = [ "!" ] ftag
      | ftag ( "=" | "!=" ) tag-value
      | ftag "=" "[" numeric-range "]"

numeric-range = [ number ] "-" [ number ]
```

Feature predicates are used in features attributes (section 6.4),
which are used in variant descriptions (section 5). Variant
descriptions can be transmitted in Alternates headers (section 8.3).

Examples of feature predicates are

    blebber, !blebber, paper=a4, colordepth=5, blex!=54,
    dpi=[300-599], colordepth=[24-]

Using the feature set of the current request, a user agent SHOULD
compute the truth value of the different feature predicates as
follows.

    ftag   true if the feature is present, false otherwise

    !ftag   true if the feature is absent, false otherwise

    ftag=V   true if the feature is present with the value V,
                false otherwise,

    ftag!=V   true if the feature is not present with the value V,
                false otherwise,

    ftag=[N-M] true if the feature is present with at least one
                numeric value, while the highest value with which it
                is present in the range N-M, false otherwise. If N
                is missing, the lower bound is 0. If M is missing,
                the upper bound is infinity.

As an example, with the feature set

    { ("blex"   , { }),
      ("colordepth" , { "5" }),
      ("UA-media"   , { "stationary" }),
      ("paper"   , { "A4", "A3" }) ,
      ("x-version" , { "104", "200" })
    }

the following predicates are true:

blex, colordepth=[4-], colordepth!=6, colordepth, !screenwidth, UA-
media=stationary, UA-media!=screen, paper=A4, paper =!A0,
colordepth=[4 - 6], x-version=[100-300], x-version=[200-300]

and the following predicates are false:

    !blex, blebber, colordepth=6, colordepth=foo, !colordepth,
    screenwidth, screenwidth=640, screenwidth!=640, x-version=99, UA-
    media=screen, paper=A0, paper=a4, x-version=[100-199], wuxta

6.4 Features attribute

     The features attribute, for which section 5.1 defines the syntax

     "{" "features" feature-list "}"

   is used in a variant description to specify how the presence or
   absence of particular feature tags in the user agent affects the
   overall quality of the variant.

     feature-list = 1%feature-list-element

     feature-list-element = (fpred | fpred-bag)
                            [";" ["+" true-improvement]
                                  ["-" false-degradation]
                            ]

     fpred-bag = "[" 1%fpred "]"

     true-improvement = short-float
     false-degradation = short-float

   Features attributes are used in variant descriptions (section 5).
   Variant descriptions can be transmitted in Alternates headers
   (section 8.3).

   Examples are:

     {features !textonly [blebber !wolx] colordepth=3;+0.7}

     {features !blink;-0.5 background;+1.5 [blebber !wolx];+1.4-0.8}

   The default value for the true-improvement is 1. The default value
   for the false-degradation is 0, or 1 if a true-improvement value is
   given.

   A user agent SHOULD, and a remote variant selection algorithm MUST
   compute the quality degradation factor associated with the features
   attribute by multiplying all quality degradation factors of the
   elements of the feature-list. Note that the result can be a factor
   greater than 1.

   A feature list element yields its true-improvement factor if the
   corresponding feature predicate is true, or if at least one element
   of the corresponding fpred-bag is true. The element yields its
   false-degradation factor otherwise.

7 Remote variant selection algorithms

   A remote variant selection algorithm is a standardized algorithm by
   which a server can choose a best variant on behalf of a negotiating
   user agent. The use of a remote algorithm can speed up the
   negotiation process by eliminating a request-response round trip.

   A remote algorithm typically computes whether the Accept- headers in
   the request contain sufficient information to allow a choice, and if
   so, which variant is the best variant. This specification does not
   define any remote algorithms, but does define a mechanism to
   negotiate on the use of such algorithms.

7.1 Version numbers

   A version numbering scheme is used to distinguish between different
   remote variant selection algorithms.

      rvsa-version = major "." minor

      major = 1*4DIGIT
      minor = 1*4DIGIT

   An algorithm with the version number X.Y, with Y>0, MUST be downwards
   compatible with all algorithms from X.0 up to X.Y. Downwards
   compatibility means that, if supplied with the same information, the
   newer algorithm MUST make the same choice, or a better choice, as the
   old algorithm. There are no compatibility requirements between
   algorithms with different major version numbers.

8 Content negotiation status codes and headers

   This specification adds one new HTTP status code, and introduces six
   new HTTP headers. It also extends the semantics of an existing
   HTTP/1.1 header.

8.1 506 Variant Also Negotiates

   The 506 status code indicates that the server has an internal
   configuration error: the chosen variant resource is configured to
   engage in transparent content negotiation itself, and is therefore
   not a proper end point in the negotiation process.

8.2 Accept-Features

   The Accept-Features request header can be used by a user agent to
   give information about the presence or absence of certain features in
   the feature set of the current request. Servers can use this
   information when running a remote variant selection algorithm.

      Note: the name 'Accept-Features' for this header was chosen
      because of symmetry considerations with other Accept- headers,
      even though the Accept-Features header will generally not contain
      an exhaustive list of features which are somehow 'accepted'. A
      more accurate name of this header would have been 'Feature-Set-
      Info'.

        Accept-Features = "Accept-Features" ":"
                    #(feature-expr *(";" feature-extension))

        feature-expr = ["!"] ftag
                     | ftag ("=" | "!=") tag-value
                     | ftag "=" "{" tag-value "}"
                     | "*"

        feature-extension = token ["=" (token | quoted-string)]

   No feature extensions are defined in this specification. An example
   is:

        Accept-Features: blex, !blebber, colordepth={5}, !screenwidth,
                  paper = A4, paper!="A2", x-version=104, *

   The different feature expressions have the following meaning:

      ftag   ftag is present

      !ftag   ftag is absent

      ftag=V   ftag is present with the value V

      ftag!=V   ftag is present, but not with the value V

      ftag={V}   ftag is present with the value V, and not with any
                  other values

      *   the expressions in this header do not fully describe
                  the feature set: feature tags not mentioned in this
                  header may also be present, and, except for the case
                  ftag={V}, tags may be present with more values than
                  mentioned.

Absence of the Accept-Features header in a request is equivalent to
the inclusion of

    Accept-Features: *

By using the Accept-Features header, a remote variant selection
algorithm can sometimes determine the truth value of a feature
predicate on behalf of the user agent. For example, with the header

    Accept-Features: blex, !blebber, colordepth={5}, !screenwidth,
              paper = A4, paper!="A2", x-version=104, *

the algorithm can determine that the following predicates are true:

    blex, colordepth=[4-], colordepth!=6, colordepth, !screenwidth,
    paper=A4, colordepth=[4-6]

and that the following predicates are false:

    !blex, blebber, colordepth=6, colordepth=foo, !colordepth,
    screenwidth, screenwidth=640, screenwidth!=640,

but the truth value of the following predicates cannot be
determined:

    UA-media=stationary, UA-media!=screen, paper!=a0,
    x-version=[100-300], x-version=[200-300], x-version=99,
    UA-media=screen, paper=A0, paper=a4, x-version=[100-199], wuxta

8.3 Alternates

   The Alternates response header is used to convey the list of variants
   bound to a negotiable resource. This list can also include
   directives for any content negotiation process. If a response from a
   transparently negotiable resource includes an Alternates header, this
   header MUST contain the complete variant list bound to the negotiable
   resource. Responses from resources which do not support transparent
   content negotiation MAY also use Alternates headers.

    Alternates = "Alternates" ":" variant-list

    variant-list = 1#(variant-description
                     | fallback-variant
                     | list-directive)

    fallback-variant = "{" <"> URI <"> "}"

    list-directive = ("proxy-rvsa" "=" <"> 0#rvsa-version <">)

```
                        | extension-list-directive

          extension-list-directive =
                          token [ "=" ( token | quoted-string ) ]
```

An example is

```
      Alternates: {"paper.1" 0.9 {type text/html} {language en}},
                  {"paper.2" 0.7 {type text/html} {language fr}},
                  {"paper.3" 1.0 {type application/postscript}
                      {language en}},
                  proxy-rvsa="1.0, 2.5"
```

Any relative URI specified in a variant-description or fallback-
variant field is relative to the request-URI. Only one fallback-
variant field may be present. If the variant selection algorithm of
the user agent finds that all described variants are unacceptable,
then it SHOULD choose the fallback variant, if present, as the best
variant. If the user agent computes the overall quality values of
the described variants, and finds that several variants share the
highest value, then the first variant with this value in the list
SHOULD be chosen as the best variant.

The proxy-rvsa directive restricts the use of remote variant
selection algorithms by proxies. If present, a proxy MUST ONLY use
algorithms which have one of the version numbers listed, or have the
same major version number and a higher minor version number as one of
the versions listed. Any restrictions set by proxy-rvsa come on top
of the restrictions set by the user agent in the Negotiate request
header. The directive proxy-rvsa="" will disable variant selection
by proxies entirely. Clients SHOULD ignore all extension-list-
directives they do not understand.

A variant list may contain multiple differing descriptions of the
same variant. This can be convenient if the variant uses conditional
rendering constructs, or if the variant resource returns multiple
representations using a multipart media type.

8.4 Negotiate

   The Negotiate request header can contain directives for any content
   negotiation process initiated by the request.

```
      Negotiate = "Negotiate" ":" 1#negotiate-directive

      negotiate-directive = "trans"
                          | "vlist"
                          | "guess-small"
```

```
                               | rvsa-version
                               | "*"
                               | negotiate-extension

      negotiate-extension = token [ "=" token ]
```

   Examples are

```
      Negotiate: 1.0, 2.5
      Negotiate: *
```

   The negotiate directives have the following meaning

      "trans"
         The user agent supports transparent content negotiation for
         the current request.

      "vlist"
         The user agent requests that any transparently negotiated
         response for the current request includes an Alternates
         header with the variant list bound to the negotiable resource.
         Implies "trans".

      "guess-small"
         The user agent allows origin servers to run a custom algorithm
         which guesses the best variant for the request, and to return
         this variant in a choice response, if the resulting choice
         response is smaller than or not much larger than a list
         response. The definition of 'not much larger' is left to
         origin server heuristics. Implies "vlist" and "trans".

      rvsa-version
         The user agent allows origin servers and proxies to run the
         remote variant selection algorithm with the indicated version
         number, or with the same major version number and a higher
         minor version number. If the algorithm has sufficient
         information to choose a best, neighboring variant, the origin
         server or proxy MAY return a choice response with this
         variant. Implies "trans".

      "*"
         The user agent allows origin servers and proxies to run any
         remote variant selection algorithm. The origin server may
         even run algorithms which have not been standardized. If the
         algorithm has sufficient information to choose a best,
         neighboring variant, the origin server or proxy MAY return a
         choice response with this variant. Implies "trans".

```

Servers SHOULD ignore all negotiate-directives they do not
understand.  If the Negotiate header allows a choice between multiple
remote variant selection algorithms which are all supported by the
server, the server SHOULD use some internal precedence heuristics to
select the best algorithm.

8.5 TCN

The TCN response header is used by a server to signal that the
resource is transparently negotiated.

```
TCN = "TCN" ":" #(response-type
                 | server-side-override-directive
                 | tcn-extension)

response-type = "list" | "choice" | "adhoc"

server-side-override-directive = "re-choose" | "keep"

tcn-extension = token ["=" (token | quoted-string)]
```

If the resource is not transparently negotiated, a TCN header MUST
NOT be included in any response.  If the resource is transparently
negotiated, a TCN header, which includes the response-type value of
the response, MUST be included in every response with a 2xx status
code or any 3xx status code, except 304, in which it MAY be included.
A TCN header MAY also be included, without a response-type value, in
other responses from transparently negotiated resources.

A server-side override directive MUST be included if the origin
server performed a server-side override when choosing the response.
If the directive is "re-choose", the server MUST include an
Alternates header with the variant bound to the negotiable resource
in the response, and user agent SHOULD use its internal variant
selection algorithm to choose, retrieve, and display the best variant
from this list.  If the directive is "keep" the user agent SHOULD NOT
renegotiate on the response, but display it directly, or act on it
directly if it is a redirection response.

Clients SHOULD ignore all tcn-extensions they do not understand.

8.6 Variant-Vary

The Variant-Vary response header can be used in a choice response to
record any vary information which applies to the variant data (the
entity body combined with some of the entity headers) contained in
the response, rather than to the response as a whole.

```
      Variant-Vary  = "Variant-Vary" ":" ( "*" | 1#field-name )
```

Use of the Variant-Vary header is discussed in section 10.2.

9 Cache validators

To allow for correct and efficient caching and revalidation of
negotiated responses, this specification extends the caching model of
HTTP/1.1 [1] in various ways.

This specification does not introduce a 'variant-list-max-age'
directive which explicitly bounds the freshness lifetime of a cached
variant list, like the 'max-age' Cache-Control directive bounds the
freshness lifetime of a cached response. However, this specification
does ensure that a variant list which is sent at a time T by the
origin server will never be re-used without revalidation by
semantically transparent caches after the time T+M. This M is the
maximum of all freshness lifetimes assigned (using max-age directives
or Expires headers) by the origin server to

    a. the responses from the negotiable resource itself, and

    b. the responses from its neighboring variant resources

If no freshness lifetimes are assigned by the origin server, M is the
maximum of the freshness lifetimes which were heuristically assigned
by all caches which can re-use the variant list.

9.1 Variant list validators

A variant list validator is an opaque value which acts as the cache
validator of a variant list bound to a negotiable resource.

```
      variant-list-validator = <quoted-string not containing any ";">
```

If two responses contain the same variant list validator, a cache can
treat the Alternates headers in these responses as equivalent (though
the headers themselves need not be identical).

9.2 Structured entity tags

A structured entity tag consists of a normal entity tag of which the
opaque string is extended with a semicolon followed by the text
(without the surrounding quotes) of a variant list validator:

```
     normal       | variant list   |   structured
   entity tag     | validator      |   entity tag
  -------------+----------------+-----------------
     "etag"       |     "vlv"      |   "etag;vlv"
    W/"etag"      |     "vlv"      |  W/"etag;vlv"
```

Note that a structured entity tag is itself also an entity tag. The
structured nature of the tag allows caching proxies capable of
transparent content negotiation to perform some optimizations defined
in section 10. When not performing such optimizations, a structured
tag SHOULD be treated as a single opaque value, according to the
general rules in HTTP/1.1. Examples of structured entity tags are:

   "xyzzy;1234" W/"xyzzy;1234" "gonkxxxx;1234" "a;b;c;;1234"

In the last example, the normal entity tag is "a;b;c;" and the
variant list validator is "1234".

If a transparently negotiated response includes an entity tag, it
MUST be a structured entity tag. The variant list validator in the
structured tag MUST act as a validator for the variant list contained
in the Alternates header. The normal entity tag in the structured
tag MUST act as a validator of the entity body in the response and of
all entity headers except Alternates.

9.3 Assigning entity tags to variants

   To allow for correct revalidation of transparently negotiated
   responses by clients, origin servers SHOULD generate all normal
   entity tags for the neighboring variant resources of the negotiable
   resource in such a way that

     1. the same tag is never used by two different variants,
        unless this tag labels exactly the same entity on all occasions,

     2. if one normal tag "X" is a prefix of another normal tag "XY",
        then "Y" must never be a semicolon followed by a variant list
        validator.

10 Content negotiation responses

   If a request on a transparently negotiated resource yields a response
   with a 2xx status code or any 3xx status code except 304, this
   response MUST always be either a list response, a choice response, or
   an adhoc response. These responses MUST always include a TCN header
   which specifies their type. Transparently negotiated responses with
   other status codes MAY also include a TCN header.

The conditions under which the different content negotiation
responses may be sent are defined in section 12.1 for origin servers
and in section 13 for proxies.

After having constructed a list, choice, or adhoc response, a server
MAY process any If-No-Match or If-Range headers in the request
message and shorten the response to a 304 (Not Modified) or 206
(Partial Content) response, following the rules in the HTTP/1.1
specification [1]. In this case, the entity tag of the shortened
response will identify it indirectly as a list, choice, or adhoc
response.

10.1 List response

A list response returns the variant list of the negotiable resource,
but no variant data. It can be generated when the server does not
want to, or is not allowed to, return a particular best variant for
the request. If the user agent supports transparent content
negotiation, the list response will cause it to select a best variant
and retrieve it.

A list response MUST contain (besides the normal headers required by
HTTP) a TCN header which specifies the "list" response-type, the
Alternates header bound to the negotiable resource, a Vary header and
(unless it was a HEAD request) an entity body which allows the user
to manually select the best variant.

An example of a list response is

```
HTTP/1.1 300 Multiple Choices
Date: Tue, 11 Jun 1996 20:02:21 GMT
TCN: list
Alternates: {"paper.1" 0.9 {type text/html} {language en}},
            {"paper.2" 0.7 {type text/html} {language fr}},
            {"paper.3" 1.0 {type application/postscript}
                {language en}}
Vary: negotiate, accept, accept-language
ETag: "blah;1234"
Cache-control: max-age=86400
Content-Type: text/html
Content-Length: 227

<h2>Multiple Choices:</h2>
<ul>
<li><a href=paper.1>HTML, English version</a>
<li><a href=paper.2>HTML, French version</a>
<li><a href=paper.3>Postscript, English version</a>
</ul>
```

   Note: A list response can have any status code, but the 300
   (Multiple Choices) code is the most appropriate one for HTTP/1.1
   clients. Some existing versions of HTTP/1.0 clients are known to
   silently ignore 300 responses, instead of handling them according
   to the HTTP/1.0 specification [2]. Servers should therefore be
   careful in sending 300 responses to non-negotiating HTTP/1.0 user
   agents, and in making these responses cacheable. The 200 (OK)
   status code can be used instead.

The Vary header in the response SHOULD ensure correct handling by
plain HTTP/1.1 caching proxies. This header can either be

   Vary: *

or a more elaborate header; see section 10.6.1.

Only the origin server may construct list responses. Depending on
the status code, a list response is cacheable unless indicated
otherwise.

According to the HTTP/1.1 specification [1], a user agent which does
not support transparent content negotiation will, when receiving a
list response with the 300 status code, display the entity body
included in the response. If the response contains a Location
header, however, the user agent MAY automatically redirect to this
location.

The handling of list responses by clients supporting transparent
content negotiation is described in sections 11.1 and 13.

10.2 Choice response

A choice response returns a representation of the best variant for
the request, and may also return the variant list of the negotiable
resource. It can be generated when the server has sufficient
information to be able to choose the best variant on behalf the user
agent, but may only be generated if this best variant is a
neighboring variant. For request from user agents which do not
support transparent content negotiation, a server may always generate
a choice response, provided that the variant returned is a
neighboring variant. The variant returned in a choice response need
not necessarily be listed in the variant list bound to the negotiable
resource.

A choice response merges a normal HTTP response from the chosen
variant, a TCN header which specifies the "choice" response-type, and
a Content-Location header giving the location of the variant.
Depending on the status code, a choice response is cacheable unless
indicated otherwise.

Origin servers and proxy caches MUST construct choice responses with
the following algorithm (or any other algorithm which gives equal end
results for the client).

In this algorithm, 'the current Alternates header' refers to the
Alternates header containing the variant list which was used to
choose the best variant, and 'the current variant list validator'
refers to the validator of this list. Section 10.4 specifies how
these two items can be obtained by a proxy cache.

The algorithm consists of four steps.

   1. Construct a HTTP request message on the best variant resource
      by rewriting the request-URI and Host header (if appropriate) of
      the received request message on the negotiable resource.

   2. Generate a valid HTTP response message, but not one with the
      304 (Not Modified) code, for the request message constructed in
      step 1.

      In a proxy cache, the response can be obtained from cache
      memory, or by passing the constructed HTTP request towards the
      origin server. If the request is passed on, the proxy MAY add,
      modify, or delete If-None-Match and If-Range headers to optimize
      the transaction with the upstream server.

         Note: the proxy should be careful not to add entity tags of
         non-neighboring variants to If-* (conditional) headers of the
         request, as there are no global uniqueness requirements for
         these tags.

   3. Only in origin servers: check for an origin server
      configuration error. If the HTTP response message generated in
      step 2 contains a TCN header, then the best variant resource is
      not a proper end point in the transparent negotiation process,
      and a 506 (Variant Also Negotiates) error response message
      SHOULD be generated instead of going to step 4.

   4. Add a number of headers to the HTTP response message generated
      in step 2.

a. Add a TCN header which specifies the "choice"
   response-type.

b. Add a Content-Location header giving the location of the
   chosen variant. Delete any Content-Location header which was
   already present.

   Note: According to the HTTP/1.1 specification [1], if the
   Content-Location header contains a relative URI, this URI
   is relative to the URI in the Content-Base header, if
   present, and relative to the request-URI if no Content-
   Base header is present.

c. If any Vary headers are present in the response message
   from step 2, add, for every Vary header, a Variant-Vary
   header with a copy of the contents of this Vary header.

d. Delete any Alternates headers which are present in in the
   response. Now, the current Alternates header MUST be added
   if this is required by the Negotiate request header, or if
   the server returns "re-choose" in the TCN response header.
   Otherwise, the current Alternates header MAY be added.

   Note: It is usually a good strategy to always add the
   current Alternates header, unless it is very large
   compared to the rest of the response.

e. Add a Vary header to ensure correct handling by plain
   HTTP/1.1 caching proxies. This header can either be

   Vary: *
   or a more elaborate header, see section 10.6.

f. To ensure compatibility with HTTP/1.0 caching proxies which
   do not recognize the Vary header, an Expires header with a
   date in the past MAY be added. See section 10.7 for more
   information.

g. If an ETag header is present in the response message from
   step 2, then extend the entity tag in that header with the
   current variant list validator, as specified in section 9.2.

   Note: Step g. is required even if the variant list itself
   is not added in step d.

h. Only in proxy caches: set the Age header of the response to

   max(variant_age , alternates_age)

where variant_age is the age of the variant response obtained
in step 2, calculated according to the rules in the HTTP/1.1
specification [1], and alternates_age is the age of the
Alternates header added in step d, calculated according to
the rules in section 10.4.

Note that a server can shorten the response produced by the above
algorithm to a 304 (Not Modified) response if an If-None-Match header
in the original request allows it. If this is the case, an
implementation of the above algorithm can avoid the unnecessary
internal construction of full response message in step 2, it need
only construct the parts which end up in the final 304 response. A
proxy cache which implements this optimization can sometimes generate
a legal 304 response even if it has not cached the variant data
itself.

An example of a choice response is:

```
HTTP/1.1 200 OK
Date: Tue, 11 Jun 1996 20:05:31 GMT
TCN: choice
Content-Type: text/html
Last-Modified: Mon, 10 Jun 1996 10:01:14 GMT
Content-Length: 5327
Cache-control: max-age=604800
Content-Location: paper.1
Alternates: {"paper.1" 0.9 {type text/html} {language en}},
            {"paper.2" 0.7 {type text/html} {language fr}},
            {"paper.3" 1.0 {type application/postscript}
                {language en}}
Etag: "gonkyyyy;1234"
Vary: negotiate, accept, accept-language
Expires: Thu, 01 Jan 1980 00:00:00 GMT

<title>A paper about ....
```

10.3 Adhoc response

An adhoc response can be sent by an origin server as an extreme
measure, to achieve compatibility with a non-negotiating or buggy
client if this compatibility cannot be achieved by sending a list or
choice response. There are very little requirements on the contents
of an adhoc response. An adhoc response MUST have a TCN header which
specifies the "adhoc" response-type, and a Vary header if the
response is cacheable. It MAY contain the Alternates header bound to
the negotiable resource.

Any Vary header in the response SHOULD ensure correct handling by
plain HTTP/1.1 caching proxies. This header can either be

     Vary: *

or a more elaborate header, see section 10.6.1. Depending on the
status code, an adhoc response is cacheable unless indicated
otherwise.

As an example of the use of an adhoc response, suppose that the
variant resource "redirect-to-blah" yields redirection (302)
responses. A choice response with this variant could look as
follows:

```
  HTTP/1.1 302 Moved Temporarily
  Date: Tue, 11 Jun 1996 20:02:28 GMT
  TCN: choice
  Content-location: redirect-to-blah
  Location: http://blah.org/
  Content-Type: text/html
  Content-Length: 62

  This document is available <a href=http://blah.org/>here</a>.
```

Suppose that the server knows that the receiving user agent has a
bug, which causes it to crash on responses which contain both a
Content-Location and a Location header. The server could then work
around this bug by performing a server-side override and sending the
following adhoc response instead:

```
     HTTP/1.1 302 Moved Temporarily
     Date: Tue, 11 Jun 1996 20:02:28 GMT
     TCN: adhoc, keep
     Location: http://blah.org/
     Content-Type: text/html
     Content-Length: 62

     This document is available <a href=http://blah.org/>here</a>.
```

10.4 Reusing the Alternates header

If a proxy cache has available a negotiated response which is
cacheable, fresh, and has ETag and Alternates headers, then it MAY
extract the Alternates header and associated variant list validator
from the response, and reuse them (without unnecessary delay) to

negotiate on behalf of the user agent (section 13) or to construct a
choice response (section 10.2). The age of the extracted Alternates
header is the age of the response from which it is extracted,
calculated according to the rules in the HTTP/1.1 specification [1].

10.5 Extracting a normal response from a choice response

   If a proxy receives a choice response, it MAY extract and cache the
   normal HTTP response contained therein. The normal response can be
   extracted by taking a copy of the choice response and then deleting
   any Content-Location, Alternates, and Vary headers, renaming any
   Variant-Vary headers to Vary headers, and shortening the structured
   entity tag in any ETag header to a normal entity tag.

   This normal response MAY be cached (as a HTTP response to the variant
   request as constructed in step 1. of section 10.2) and reused to
   answer future direct requests on the variant resource, according to
   the rules in the HTTP/1.1 specification [1].

      Note: The caching of extracted responses can decrease the upstream
      bandwidth usage with up to a factor 2, because two independent
      HTTP/1.1 cache entries, one associated with the negotiable
      resource URI and one with the variant URI, are created in the same
      transaction. Without this optimization, both HTTP/1.1 cache
      entries can only be created by transmitting the variant data
      twice.

   For security reasons (see section 14.2), an extracted normal response
   MUST NEVER be cached if belongs to a non-neighboring variant
   resource. If the choice response claims to contain data for a non-
   neighboring variant resource, the proxy SHOULD reject the choice
   response as a probable spoofing attempt.

10.6 Elaborate Vary headers

   If a HTTP/1.1 [1] server can generate varying responses for a request
   on some resource, then the server MUST include a Vary header in these
   responses if they are cacheable. This Vary header is a signal to
   HTTP/1.1 caches that something special is going on. It prevents the
   caches from returning the currently chosen response for every future
   request on the resource.

   Servers engaging in transparent content negotiation will generate
   varying responses. Therefore, cacheable list, choice, and adhoc
   responses MUST always include a Vary header.

The most simple Vary header which can be included is

        Vary: *

This header leaves the way in which the response is selected by the
server completely unspecified.

A more elaborate Vary header MAY be used to allow for certain
optimizations in HTTP/1.1 caches which do not have specific
optimizations for transparent content negotiation, but which do cache
multiple variant responses for one resource. Such a more elaborate
Vary header lists all request headers which can be used by the server
when selecting a response for a request on the resource.

10.6.1 Construction of an elaborate Vary header

   Origin servers can construct a more elaborate Vary header in the
   following way. First, start with the header

        Vary: negotiate

'negotiate' is always included because servers use the information in
the Negotiate header when choosing between a list, choice, or adhoc
response.

Then, if any of the following attributes is present in any variant
description in the Alternates header, add the corresponding header
name to the Vary header

           attribute | header name to add
           -----------+---------------------
            type | accept
            charset | accept-charset
            language | accept-language
            features | accept-features

The Vary header constructed in this way specifies the response
variation which can be caused by the use of a variant selection
algorithm in proxies. If the origin server will in some cases, for
example if contacted by a non-negotiating user agent, use a custom
negotiation algorithm which takes additional headers into account,
these names of these headers SHOULD also be added to the Vary header.

10.6.2 Caching of an elaborate Vary header

   A proxy cache cannot construct an elaborate vary header using the
   method above, because this method requires exact knowledge of any
   custom algorithms present in the origin server. However, when
   extracting an Alternates header from a response (section 10.4) caches
   MAY also extract the Vary header in the response, and reuse it along
   with the Alternates header. A clean Vary header can however only be
   extracted if the variant does not vary itself, i.e. if a Variant-Vary
   header is absent.

10.7 Adding an Expires header for HTTP/1.0 compatibility

   To ensure compatibility with HTTP/1.0 caching proxies which do not
   recognize the Vary header, an Expires header with a date in the past
   can be added to the response, for example

        Expires: Thu, 01 Jan 1980 00:00:00 GMT

   If this is done by an origin server, the server SHOULD usually also
   include a Cache-Control header for the benefit of HTTP/1.1 caches,
   for example

            Cache-Control: max-age=604800

   which overrides the freshness lifetime of zero seconds specified by
   the included Expires header.

      Note: This specification only claims downwards compatibility with
      the HTTP/1.0 proxy caches which implement the HTTP/1.0
      specification [2]. Some legacy proxy caches which return the
      HTTP/1.0 protocol version number do not honor the HTTP/1.0 Expires
      header as specified in [2]. Methods for achieving compatibility
      with such proxy caches are beyond the scope of this specification.

10.8 Negotiation on content encoding

   Negotiation on the content encoding of a response is orthogonal to
   transparent content negotiation. The rules for when a content
   encoding may be applied are the same as in HTTP/1.1: servers MAY
   content-encode responses that are the result of transparent content
   negotiation whenever an Accept-Encoding header in the request allows
   it. When negotiating on the content encoding of a cacheable
   response, servers MUST add the accept-encoding header name to the
   Vary header of the response, or add 'Vary: *'.

Servers SHOULD always be able to provide unencoded versions of every
transparently negotiated response. This means in particular that
every variant in the variant list SHOULD at least be available in an
unencoded form.

Like HTTP/1.1, this specification allows proxies to encode or decode
relayed or cached responses on the fly, unless explicitly forbidden
by a Cache-Control directive. The encoded or decoded response still
contains the same variant as far as transparent content negotiation
is concerned. Note that HTTP/1.1 requires proxies to add a Warning
header if the encoding of a response is changed.

11 User agent support for transparent negotiation

This section specifies the requirements a user agent needs to satisfy
in order to support transparent negotiation. If the user agent
contains an internal cache, this cache MUST conform to the rules for
proxy caches in section 13.

11.1 Handling of responses

If a list response is received when a resource is accessed, the user
agent MUST be able to automatically choose, retrieve, and display the
best variant, or display an error message if none of the variants are
acceptable.

If a choice response is received when a resource is accessed, the
usual action is to automatically display the enclosed entity.
However, if a remote variant selection algorithm which was enabled
could have made a choice different from the choice the local
algorithm would make, the user agent MAY apply its local algorithm to
any variant list in the response, and automatically retrieve and
display another variant if the local algorithm makes an other choice.

When receiving a choice response, a user agent SHOULD check if
variant resource is a neighboring variant resource of the negotiable
resource. If this is not the case, the user agent SHOULD reject the
choice response as a probable spoofing attempt and display an error
message, for example by internally replacing the choice response with
a 502 (bad gateway) response.

11.2 Presentation of a transparently negotiated resource

If the user agent is displaying a variant which is not an embedded or
inlined object and which is the result of transparent content
negotiation, the following requirements apply.

   1. The user agent SHOULD allow the user to review a list of all
      variants bound to the negotiable resource, and to manually
      retrieve another variant if desired. There are two general ways
      of providing such a list. First, the information in the
      Alternates header of the negotiable resource could be used to
      make an annotated menu of variants. Second, the entity included
      in a list response of the negotiable resource could be displayed.
      Note that a list response can be obtained by doing a GET request
      which only has the "trans" directive in the Negotiate header.

   2. The user agent SHOULD make available though its user interface
      some indication that the resource being displayed is a negotiated
      resource instead of a plain resource. It SHOULD also allow the
      user to examine the variant list included in the Alternates
      header. Such a notification and review mechanism is needed
      because of privacy considerations, see section 14.1.

   3. If the user agent shows the URI of the displayed information to
      the user, it SHOULD be the negotiable resource URI, not the
      variant URI that is shown. This encourages third parties, who
      want to refer to the displayed information in their own
      documents, to make a hyperlink to the negotiable resource as a
      whole, rather than to the variant resource which happens to be
      shown. Such correct linking is vital for the interoperability of
      content across sites. The user agent SHOULD however also provide
      a means for reviewing the URI of the particular variant which is
      currently being displayed.

   4. Similarly, if the user agent stores a reference to the
      displayed information for future use, for example in a hotlist,
      it SHOULD store the negotiable resource URI, not the variant URI.

   It is encouraged, but not required, that some of the above
   functionality is also made available for inlined or embedded objects,
   and when a variant which was selected manually is being displayed.

12 Origin server support for transparent negotiation

12.1 Requirements

   To implement transparent negotiation on a resource, the origin server
   MUST be able to send a list response when getting a GET request on
   the resource. It SHOULD also be able to send appropriate list
   responses for HEAD requests. When getting a request on a
   transparently negotiable resource, the origin server MUST NEVER
   return a response with a 2xx status code or any 3xx status code,
   except 304, which is not a list, choice, or adhoc response.

If the request includes a Negotiate header with a "vlist" or "trans"
directive, but without any directive which allows the server to
select a best variant, a list response MUST ALWAYS be sent, except
when the server is performing a server-side override for bug
compatibility. If the request includes a Negotiate header with a
"vlist" or "guess-small" directive, an Alternates header with the
variant list bound to the negotiable resource MUST ALWAYS be sent in
any list, choice, or adhoc response, except when the server is
performing a server-side override for bug compatibility.

If the Negotiate header allows it, the origin server MAY run a remote
variant selection algorithm. If the algorithm has sufficient
information to choose a best variant, and if the best variant is a
neighboring variant, the origin server MAY return a choice response
with this variant.

When getting a request on a transparently negotiable resource from a
user agent which does not support transparent content negotiation,
the origin server MAY use a custom algorithm to select between
sending a list, choice, or adhoc response.

The following table summarizes the rules above.

Req on trans neg resource?	Usr agnt capable of TCN?	server-side override?	Response may be:				
			list	choice	adhoc	normal	error
Yes	Yes	No	always	smt(*)	never	never	always
Yes	Yes	Yes	always	always	always	never	always
Yes	No	-	always	always	always	never	always
No	-	-	never	never	never	always	always

      (*) sometimes, when allowed by the Negotiate request header

Negotiability is a binary property: a resource is either
transparently negotiated, or it is not. Origin servers SHOULD NOT
vary the negotiability of a resource, or the variant list bound to
that resource, based on the request headers which are received. The
variant list and the property of being negotiated MAY however change
through time. The Cache-Control header can be used to control the
propagation of such time-dependent changes through caches.

It is the responsibility of the author of the negotiable resource to
ensure that all resources in the variant list serve the intended
content, and that the variant resources do not engage in transparent

   content negotiation themselves.

12.2 Negotiation on transactions other than GET and HEAD

   If a resource is transparently negotiable, this only has an impact on
   the GET and HEAD transactions on the resource. It is not possible
   (under this specification) to do transparent content negotiation on
   the direct result of a POST request.

   However, a POST request can return an unnegotiated 303 (See Other)
   response which causes the user agent to do a GET request on a second
   resource. This second resource could then use transparent content
   negotiation to return an appropriate final response. The figure
   below illustrates this.

```
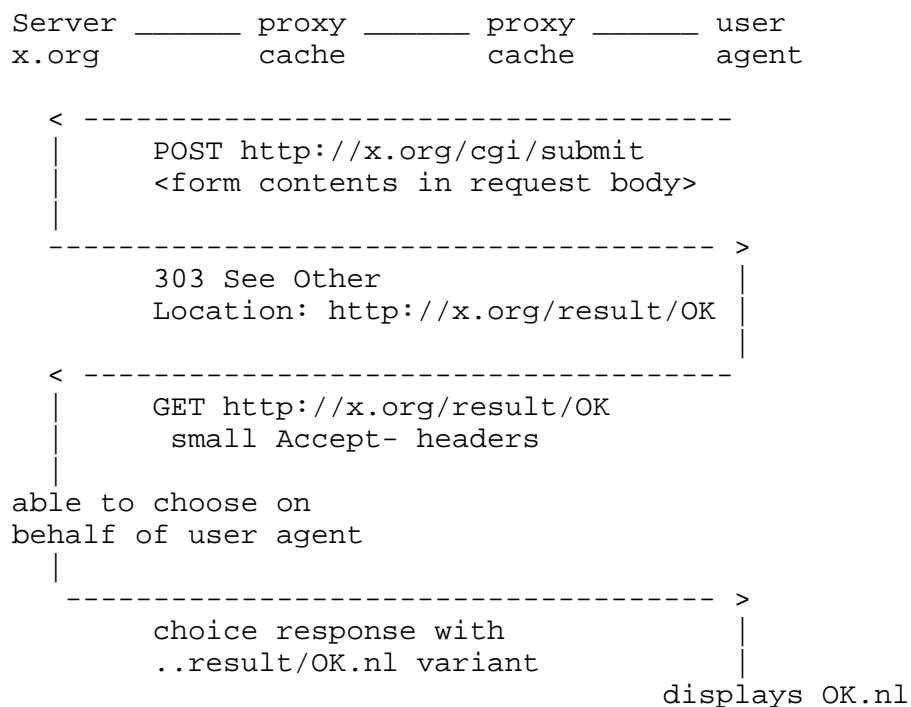      Server _____ proxy _____ proxy _____ user
      x.org           cache         cache         agent

        < ------------------------------------
        |       POST http://x.org/cgi/submit
        |       <form contents in request body>
        |
        ------------------------------------- >
                303 See Other                |
                Location: http://x.org/result/OK |
                                             |
        < ------------------------------------
        |       GET http://x.org/result/OK
        |        small Accept- headers
        |
      able to choose on
      behalf of user agent
         |
         ------------------------------------- >
                 choice response with          |
                 ..result/OK.nl variant        |
                                          displays OK.nl
```

   See the HTTP/1.1 specification [1] for details on the 303 (See Other)
   status code. Note that this status code is not understood by some
   HTTP/1.0 clients.

13 Proxy support for transparent negotiation

   Transparent content negotiation is an extension on top of HTTP/1.x.
   It is designed to work through any proxy which only implements the
   HTTP/1.1 specification [1]. If Expires headers are added as
   discussed in section 10.7, negotiation will also work though proxies

which implement HTTP/1.0 [2]. Thus, every HTTP/1.0 or HTTP/1.1 proxy
provides support for transparent content negotiation. However, if it
is to be claimed that a HTTP/1.x proxy offers transparent content
negotiation services, at least one of the specific optimizations
below MUST be implemented.

An HTTP/1.x proxy MUST ONLY optimize (change) the HTTP traffic
flowing through it in ways which are explicitly allowed by the
specification(s) it conforms to. A proxy which supports transparent
content negotiation on top of HTTP/1.x MAY perform the optimizations
allowed for by HTTP/1.x. In addition, it MAY perform three
additional optimizations, defined below, on the HTTP traffic for
transparently negotiated resources and their neighboring variant
resources.

First, when getting a request on a transparently negotiable resource
from a user agent which supports transparent content negotiation, the
proxy MAY return any cached, fresh list response from that resource,
even if the selecting request headers, as specified by the Vary
header, do not match.

Second, when allowed by the user agent and origin server, a proxy MAY
reuse an Alternates header taken from a previous response (section
10.4) to run a remote variant selection algorithm. If the algorithm
has sufficient information to choose a best variant, and if the best
variant is a neighboring variant, the proxy MAY return a choice
response with this variant.

Third, if a proxy receives a choice response, it MAY extract and
cache the normal response embedded therein, as described in section
10.5.

14 Security and privacy considerations

14.1 Accept- headers revealing personal information

   Accept- headers, in particular Accept-Language headers, may reveal
   information which the user would rather keep private unless it will
   directly improve the quality of service. For example, a user may not
   want to send language preferences to sites which do not offer multi-
   lingual content. The transparent content negotiation mechanism
   allows user agents to omit sending of the Accept-Language header by
   default, without adversely affecting the outcome of the negotiation
   process if transparently negotiated multi-lingual content is
   accessed.

However, even if Accept- headers are never sent, the automatic
selection and retrieval of a variant by a user agent will reveal a
preference for this variant to the server. A malicious service
author could provide a page with 'fake' negotiability on (ethnicity-
correlated) languages, with all variants actually being the same
English document, as a means of obtaining privacy-sensitive
information. Such a plot would however be visible to an alert victim
if the list of available variants and their properties is reviewed.

Some additional privacy considerations connected to Accept- headers
are discussed in [1].

14.2 Spoofing of responses from variant resources

The caching optimization in section 10.5 gives the implementer of a
negotiable resource control over the responses cached for all
neighboring variant resources. This is a security problem if a
neighboring variant resource belongs to another author. To provide
security in this case, the HTTP server will have to filter the
Content-Location headers in the choice responses generated by the
negotiable resource implementation.

14.3 Security holes revealed by negotiation

Malicious servers could use transparent content negotiation as a
means of obtaining information about security holes which may be
present in user agents. This is a risk in particular for negotiation
on the availability of scripting languages and libraries.

15 Internationalization considerations

This protocol defines negotiation facilities which can be used for
the internationalization of web content. For the
internationalization of list response bodies (section 10.1), HTTP/1.0
style negotiation (section 4.2) can be used.

16 Acknowledgments

Work on HTTP content negotiation has been done since at least 1993.
The authors are unable to trace the origin of many of the ideas
incorporated in this document. Many members of the HTTP working
group have contributed to the negotiation model in this
specification. The authors wish to thank the individuals who have
commented on earlier versions of this document, including Brian
Behlendorf, Daniel DuBois, Martin J. Duerst, Roy T. Fielding, Jim
Gettys, Yaron Goland, Dirk van Gulik, Ted Hardie, Graham Klyne, Scott
Lawrence, Larry Masinter, Jeffrey Mogul, Henrik Frystyk Nielsen,
Frederick G.M. Roeber, Paul Sutton, and Klaus Weide and Mark Wood.

17 References

   [1] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., and
       T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC
       2068, January 1997.

   [2] Berners-Lee, T., Fielding, R., and H. Frystyk, "Hypertext
       Transfer Protocol -- HTTP/1.0", RFC 1945, May 1996.

   [3] Holtman, K., and A. Mutz, "HTTP Remote Variant Selection
       Algorithm -- RVSA/1.0", RFC 2296, March 1998.

   [4] Bradner, S., "Key words for use in RFCs to Indicate Requirement
       Levels", BCP 14, RFC 2119, March 1997.

   [5] Yergeau, F., "UTF-8, a transformation format of Unicode and ISO
       10646", RFC 2044, October 1996.

18 Authors' Addresses

   Koen Holtman
   Technische Universiteit Eindhoven
   Postbus 513
   Kamer HG 6.57
   5600 MB Eindhoven (The Netherlands)

   EMail: koen@win.tue.nl

   Andrew H. Mutz
   Hewlett-Packard Company
   1501 Page Mill Road 3U-3
   Palo Alto CA 94304, USA

   Fax +1 415 857 4691
   EMail: mutz@hpl.hp.com

19 Appendix: Example of a local variant selection algorithm

   A negotiating user agent will choose the best variant from a variant
   list with a local variant selection algorithm. This appendix
   contains an example of such an algorithm.

   The inputs of the algorithm are a variant list from an Alternates
   header, and an agent-side configuration database, which contains

      - the feature set of the current request,

      - a collection of quality values assigned to media types,
        languages, and charsets for the current request, following the
        model of the corresponding HTTP/1.1 [1] Accept- headers,

      - a table which lists 'forbidden' combinations of media types and
        charsets, i.e. combinations which cannot be displayed because of
        some internal user agent limitation.

   The output of the algorithm is either the best variant, or the
   conclusion that none of the variants are acceptable.

19.1 Computing overall quality values

   As a first step in the local variant selection algorithm, the overall
   qualities associated with all variant descriptions in the list are
   computed.

   The overall quality Q of a variant description is the value

      $Q = round5(qs * qt * qc * ql * qf * qa)$

   where rounds5 is a function which rounds a floating point value to 5
   decimal places after the point. It is assumed that the user agent
   can run on multiple platforms: the rounding function makes the
   algorithm independent of the exact characteristics of the underlying
   floating point hardware.

   The factors qs, qt, qc, ql, qf, and qa are determined as follows.

      qs Is the source quality factor in the variant description.

      qt The media type quality factor is 1 if there is no type
         attribute in the variant description. Otherwise, it is the
         quality value assigned to this type by the configuration
         database. If the database does not assign a value, then the
         factor is 0.

   qc The charset quality factor is 1 if there is no charset
      attribute in the variant description. Otherwise, it is the
      quality value assigned to this charset by the configuration
      database. If the database does not assign a value, then the
      factor is 0.

   ql The language quality factor is 1 if there is no language
      attribute in the variant description. Otherwise, it is the
      highest quality value the configuration database assigns to any
      of the languages listed in the language attribute. If the
      database does not assign a value to any of the languages
      listed, then the factor is 0.

   qf The features quality factor is 1 if there is no features
      attribute in the variant description. Otherwise, it is the
      quality degradation factor computed for the features attribute
      using the feature set of the current request.

   qa The quality adjustment factor is 0 if the variant description
      lists a media type - charset combination which is 'forbidden'
      by the table, and 1 otherwise.

As an example, if a variant list contains the variant description

   {"paper.2" 0.7 {type text/html} {language fr}}

and if the configuration database contains the quality value
assignments

   types: text/html;q=1.0, type application/postscript;q=0.8
   languages: en;q=1.0, fr;q=0.5

then the local variant selection algorithm will compute the overall
quality for the variant description as follows:

   {"paper.2" 0.7 {type text/html} {language fr}}
                   | | |
                   | | |
                   | | |
                   V   V   V
     round5 (0.7   *   1.0   *   0.5) = 0.35000

With same configuration database, the variant list

   {"paper.1" 0.9 {type text/html} {language en}},
   {"paper.2" 0.7 {type text/html} {language fr}},
   {"paper.3" 1.0 {type application/postscript} {language en}}

would yield the following computations:

```
      round5 ( qs  * qt  * qc  * ql  * qf  * qa ) = Q
              ---   ---   ---   ---   ---   ---
   paper.1:  0.9 * 1.0 * 1.0 * 1.0 * 1.0 * 1.0  = 0.90000
   paper.1:  0.7 * 1.0 * 1.0 * 0.5 * 1.0 * 1.0  = 0.35000
   paper.3:  1.0 * 0.8 * 1.0 * 1.0 * 1.0 * 1.0  = 0.80000
```

19.2 Determining the result

   Using all computed overall quality values, the end result of the
   local variant selection algorithm is determined as follows.

   If all overall quality values are 0, then the best variant is the
   fallback variant, if there is one in the list, else the result is the
   conclusion that none of the variants are acceptable.

   If at least one overall quality value is greater than 0, then the
   best variant is the variant which has the description with the
   highest overall quality value, or, if there are multiple variant
   descriptions which share the highest overall quality value, the
   variant of the first variant description in the list which has this
   highest overall quality value.

19.3 Ranking dimensions

   Consider the following variant list:

      {"paper.greek"   1.0 {language el} {charset ISO-8859-7}},
      {"paper.english" 1.0 {language en} {charset ISO-8859-1}}

   It could be the case that the user prefers the language "el" over
   "en", while the user agent can render "ISO-8859-1" better than "ISO-
   8859-7". The result is that in the language dimension, the first
   variant is best, while the second variant is best in the charset
   dimension. In this situation, it would be preferable to choose the
   first variant as the best variant: the user settings in the language
   dimension should take precedence over the hard-coded values in the
   charset dimension.

   To express this ranking between dimensions, the user agent
   configuration database should have a higher spread in the quality
   values for the language dimension than for the charset dimension.
   For example, with

      languages: el;q=1.0, en-gb;q=0.7, en;q=0.6, da;q=0, ...

      charsets: ISO-8859-1;q=1.0, ISO-8859-7;q=0.95,
                 ISO-8859-5;q=0.97, unicode-1-1;q=0, ...

the first variant will have an overall quality of 0.95000, while the
second variant will have an overall quality 0.70000. This makes the
first variant the best variant.

20 Appendix: feature negotiation examples

This appendix contains examples of the use of feature tags in variant
descriptions. The tag names used here are examples only, they do not
in general reflect the tag naming scheme proposed in [4].

20.1 Use of feature tags

Feature tags can be used in variant lists to express the quality
degradation associated with the presence or absence of certain
features. One example is

     {"index.html.plain" 0.7 },
     {"index.html"   1.0 {features tables frames}}

Here, the "{features tables frames}" part expresses that index.html
uses the features tagged as tables and frames. If these features are
absent, the overall quality of index.html degrades to 0. Another
example is

     {"home.graphics" 1.0 {features !textonly}},
     {"home.textonly" 0.7 }

where the "{features !textonly}" part expresses that home.graphics
requires the absence of the textonly feature. If the feature is
present, the overall quality of home.graphics degrades to 0.

The absence of a feature need not always degrade the overall quality
to 0. In the example

     {"x.html.1" 1.0 {features fonts;-0.7}}

the absence of the fonts feature degrades the quality with a factor
of 0.7. Finally, in the example

      {"y.html" 1.0 {features [blebber wolx] }}

The "[blebber wolx]" expresses that y.html requires the presence of
the blebber feature or the wolx feature. This construct can be used
in a number of cases:

   1. blebber and wolx actually tag the same feature, but they were
      registered by different people, and some user agents say they
      support blebber while others say they support wolx.

   2. blebber and wolx are HTML tags of different vendors which
      implement the same functionality, and which are used together in
      y.html without interference.

   3. blebber and wolx are HTML tags of different vendors which
      implement the same functionality, and y.html uses the tags in a
      conditional HTML construct.

   4. blebber is a complicated HTML tag with only a sketchy
      definition, implemented by one user agent vendor, and wolx
      indicates implementation of a well-defined subset of the blebber
      tag by some other vendor(s). y.html uses only this well-defined
      subset.

20.2 Use of numeric feature tags

   As an example of negotiation in a numeric area, the following variant
   list describes four variants with title graphics designed for
   increasing screen widths:

      {"home.pda"   1.0 {features screenwidth=[-199] }},
      {"home.narrow" 1.0 {features screenwidth=[200-599] }},
      {"home.normal" 1.0 {features screenwidth=[600-999] }},
      {"home.wide"   1.0 {features screenwidth=[1000-] }},
      {"home.normal"}

   The last element of the list specifies a safe default for user agents
   which do not implement screen width negotiation. Such user agents
   will reject the first four variants as unusable, as they seem to rely
   on a feature which they do not understand.

20.3 Feature tag design

   When designing a new feature tag, it is important to take into
   account that existing user agents, which do not recognize the new tag
   will treat the feature as absent. In general, a new feature tag
   needs to be designed in such a way that absence of the tag is the
   default case which reflects current practice. If this design
   principle is ignored, the resulting feature tag will generally be
   unusable.

   As an example, one could try to support negotiation between
   monochrome and color content by introducing a 'color' feature tag,
   the presence of which would indicate the capability to display color
   graphics. However, if this new tag is used in a variant list, for
   example

      {"rainbow.gif"   1.0 {features color} }

```
{"rainbow.mono.gif" 0.6 {features !color}}
```

then existing user agents, which would not recognize the color tag,
would all display the monochrome rainbow. The color tag is therefore
unusable in situations where optimal results for existing user agents
are desired. To provide for negotiation in this area, one must
introduce a 'monochrome' feature tag; its presence indicates that the
user agent can only render (or the user prefers to view) monochrome
graphics.

21 Appendix: origin server implementation considerations

21.1 Implementation with a CGI script

Transparent content negotiation has been designed to allow a broad
range of implementation options at the origin server side. A very
minimal implementation can be done using the CGI interface. The CGI
script below is an example.

```
#!/bin/sh

cat - <<'blex'
TCN: list
Alternates: {"stats.tables.html" 1.0 {type text/html} {features
tables}}, {"stats.html" 0.8 {type text/html}}, {"stats.ps" 0.95
{type application/postscript}}
Vary: *
Content-Type: text/html

<title>Multiple Choices for Web Statistics</title>
<h2>Multiple Choices for Web Statistics:</h2>
<ul>
<li><a href=stats.tables.html>Version with HTML tables</a>
<p>
<li><a href=stats.html>Version without HTML tables</a>
<p>
<li><a href=stats.ps>Postscript version</a>
</ul>
blex
```

The Alternates header in the above script must be read as a single
line. The script always generates a list response with the 200 (OK)
code, which ensures compatibility with non-negotiating HTTP/1.0
agents.

21.2 Direct support by HTTP servers

   Sophisticated HTTP servers could make a transparent negotiation
   module available to content authors. Such a module could incorporate
   a remote variant selection algorithm and an implementation of the
   algorithm for generating choice responses (section 10.2). The
   definition of interfaces to such modules is beyond the scope of this
   specification.

21.3 Web publishing tools

   Web publishing tools could automatically generate several variants of
   a document (for example the original TeX version, a HTML version with
   tables, a HTML version without tables, and a Postscript version),
   together with an appropriate variant list in the interface format of
   a HTTP server transparent negotiation module. This would allow
   documents to be published as transparently negotiable resources.

22 Appendix: Example of choice response construction

   The following is an example of the construction of a choice response
   by a proxy cache which supports HTTP/1.1 and transparent content
   negotiation. The use of the HTTP/1.1 conditional request mechanisms
   is also shown.

   Assume that a user agent has cached a variant list with the validator
   "1234" for the negotiable resource http://x.org/paper. Also assume
   that it has cached responses from two neighboring variants, with the
   entity tags "gonkyyyy" and W/"a;b". Assume that all three user agent
   cache entries are stale: they would need to be revalidated before the
   user agent can use them. If http://x.org/paper accessed in this
   situation, the user agent could send the following request to its
   proxy cache:

```
     GET /paper HTTP/1.1
     Host: x.org
     User-Agent: WuxtaWeb/2.4
     Negotiate: 1.0
     Accept: text/html, application/postscript;q=0.4, */*
     Accept-Language: en
     If-None-Match: "gonkyyyy;1234", W/"a;b;1234"
```

   Assume that the proxy cache has cached the same three items as the
   user agent, but that it has revalidated the variant list 8000 seconds
   ago, so that the list is still fresh for the proxy. This means that
   the proxy can run a remote variant selection algorithm on the list
   and the incoming request.

Assume that the remote algorithm is able to choose paper.html.en as
the best variant. The proxy can now construct a choice response,
using the algorithm in section 10.2. In steps 1 and 2 of the
algorithm, the proxy can construct the following conditional request
on the best variant, and send it to the origin server:

```
GET /paper.html.en HTTP/1.1
Host: x.org
User-Agent: WuxtaWeb/2.4
Negotiate: 1.0
Accept: text/html, application/postscript;q=0.4, */*
Accept-Language: en
If-None-Match: "gonkyyyy", W/"a;b"
Via: 1.1 fred
```

On receipt of the response

```
HTTP/1.1 304 Not Modified
Date: Tue, 11 Jun 1996 20:05:31 GMT
Etag: "gonkyyyy"
```

from the origin server, the proxy can use its freshly revalidated
paper.html.en cache entry to expand the response to a non-304
response:

```
HTTP/1.1 200 OK
Date: Tue, 11 Jun 1996 20:05:31 GMT
Content-Type: text/html
Last-Modified: Mon, 10 Jun 1996 10:01:14 GMT
Content-Length: 5327
Cache-control: max-age=604800
Etag: "gonkyyyy"
Via: 1.1 fred
Age: 0

<title>A paper about ....
```

Using this 200 response, the proxy can construct a choice response
in step 4 of the algorithm:

```
HTTP/1.1 200 OK
Date: Tue, 11 Jun 1996 20:05:31 GMT
TCN: choice
Content-Type: text/html
Last-Modified: Mon, 10 Jun 1996 10:01:14 GMT
Content-Length: 5327
Cache-control: max-age=604800
Content-Location: paper.html.en
```

```
   Alternates: {"paper.html.en" 0.9 {type text/html} {language en}},
               {"paper.html.fr" 0.7 {type text/html} {language fr}},
               {"paper.ps.en"   1.0 {type application/postscript}
                   {language en}}

   Etag: "gonkyyyy;1234"
   Vary: negotiate, accept, accept-language
   Expires: Thu, 01 Jan 1980 00:00:00 GMT
   Via: 1.1 fred
   Age: 8000

   <title>A paper about ....
```

   The choice response can subsequently be shortened to a 304 response,
   because of the If-None-Match header in the original request from the
   user agent. Thus, the proxy can finally return

```
   HTTP/1.1 304 Not Modified
   Date: Tue, 11 Jun 1996 20:05:31 GMT
   Etag: "gonkyyyy;1234"
   Content-Location: paper.html.en
   Vary: negotiate, accept, accept-language
   Expires: Thu, 01 Jan 1980 00:00:00 GMT
   Via: 1.1 fred
   Age: 8000
```

   to the user agent.

23 Full Copyright Statement