                Relative Uniform Resource Locators

Status of this Memo

   This document specifies an Internet standards track protocol for the
   Internet community, and requests discussion and suggestions for
   improvements. Please refer to the current edition of the "Internet
   Official Protocol Standards" (STD 1) for the standardization state
   and status of this protocol. Distribution of this memo is unlimited.

Abstract

   A Uniform Resource Locator (URL) is a compact representation of the
   location and access method for a resource available via the Internet.
   When embedded within a base document, a URL in its absolute form may
   contain a great deal of information which is already known from the
   context of that base document's retrieval, including the scheme,
   network location, and parts of the url-path. In situations where the
   base URL is well-defined and known to the parser (human or machine),
   it is useful to be able to embed URL references which inherit that
   context rather than re-specifying it in every instance. This
   document defines the syntax and semantics for such Relative Uniform
   Resource Locators.

1. Introduction

   This document describes the syntax and semantics for "relative"
   Uniform Resource Locators (relative URLs): a compact representation
   of the location of a resource relative to an absolute base URL. It
   is a companion to RFC 1738, "Uniform Resource Locators (URL)" [2],
   which specifies the syntax and semantics of absolute URLs.

   A common use for Uniform Resource Locators is to embed them within a
   document (referred to as the "base" document) for the purpose of
   identifying other Internet-accessible resources. For example, in
   hypertext documents, URLs can be used as the identifiers for
   hypertext link destinations.

   Absolute URLs contain a great deal of information which may already
   be known from the context of the base document's retrieval, including
   the scheme, network location, and parts of the URL path. In
   situations where the base URL is well-defined and known, it is useful
   to be able to embed a URL reference which inherits that context

rather than re-specifying it within each instance. Relative URLs can
also be used within data-entry dialogs to decrease the number of
characters necessary to describe a location.

In addition, it is often the case that a group or "tree" of documents
has been constructed to serve a common purpose; the vast majority of
URLs in these documents point to locations within the tree rather
than outside of it. Similarly, documents located at a particular
Internet site are much more likely to refer to other resources at
that site than to resources at remote sites.

Relative addressing of URLs allows document trees to be partially
independent of their location and access scheme. For instance, it is
possible for a single set of hypertext documents to be simultaneously
accessible and traversable via each of the "file", "http", and "ftp"
schemes if the documents refer to each other using relative URLs.
Furthermore, document trees can be moved, as a whole, without
changing any of the embedded URLs. Experience within the World-Wide
Web has demonstrated that the ability to perform relative referencing
is necessary for the long-term usability of embedded URLs.

2. Relative URL Syntax

The syntax for relative URLs is a shortened form of that for absolute
URLs [2], where some prefix of the URL is missing and certain path
components ("." and "..") have a special meaning when interpreting a
relative path. Because a relative URL may appear in any context that
could hold an absolute URL, systems that support relative URLs must
be able to recognize them as part of the URL parsing process.

Although this document does not seek to define the overall URL
syntax, some discussion of it is necessary in order to describe the
parsing of relative URLs. In particular, base documents can only
make use of relative URLs when their base URL fits within the
generic-RL syntax described below. Although some URL schemes do not
require this generic-RL syntax, it is assumed that any document which
contains a relative reference does have a base URL that obeys the
syntax. In other words, relative URLs cannot be used within
documents that have unsuitable base URLs.

2.1. URL Syntactic Components

The URL syntax is dependent upon the scheme. Some schemes use
reserved characters like "?" and ";" to indicate special components,
while others just consider them to be part of the path. However,
there is enough uniformity in the use of URLs to allow a parser to
resolve relative URLs based upon a single, generic-RL syntax. This
generic-RL syntax consists of six components:

```
<scheme>://<net_loc>/<path>;<params>?<query>#<fragment>
```

each of which, except <scheme>, may be absent from a particular URL.
These components are defined as follows (a complete BNF is provided
in Section 2.2):

    scheme ":"   ::= scheme name, as per Section 2.1 of RFC 1738 [2].

    "//" net_loc ::= network location and login information, as per
                     Section 3.1 of RFC 1738 [2].

    "/" path   ::= URL path, as per Section 3.1 of RFC 1738 [2].

    ";" params   ::= object parameters (e.g., ";type=a" as in
                     Section 3.2.2 of RFC 1738 [2]).

    "?" query   ::= query information, as per Section 3.3 of
                     RFC 1738 [2].

    "#" fragment ::= fragment identifier.

Note that the fragment identifier (and the "#" that precedes it) is
not considered part of the URL. However, since it is commonly used
within the same string context as a URL, a parser must be able to
recognize the fragment when it is present and set it aside as part of
the parsing process.

The order of the components is important. If both <params> and
<query> are present, the <query> information must occur after the
<params>.

2.2. BNF for Relative URLs

   This is a BNF-like description of the Relative Uniform Resource
   Locator syntax, using the conventions of RFC 822 [5], except that "|"
   is used to designate alternatives. Briefly, literals are quoted with
   "", parentheses "(" and ")" are used to group elements, optional
   elements are enclosed in [brackets], and elements may be preceded
   with <n>* to designate n or more repetitions of the following
   element; n defaults to 0.

   This BNF also describes the generic-RL syntax for valid base URLs.
   Note that this differs from the URL syntax defined in RFC 1738 [2] in
   that all schemes are required to use a single set of reserved
   characters and use them consistently within the major URL components.

```
   URL        = ( absoluteURL | relativeURL ) [ "#" fragment ]

   absoluteURL = generic-RL | ( scheme ":" *( uchar | reserved ) )

   generic-RL  = scheme ":" relativeURL

   relativeURL = net_path | abs_path | rel_path

   net_path    = "//" net_loc [ abs_path ]
   abs_path    = "/"  rel_path
   rel_path    = [ path ] [ ";" params ] [ "?" query ]

   path        = fsegment *( "/" segment )
   fsegment    = 1*pchar
   segment     =  *pchar

   params      = param *( ";" param )
   param       = *( pchar | "/" )

   scheme      = 1*( alpha | digit | "+" | "-" | "." )
   net_loc     =  *( pchar | ";" | "?" )
   query       =  *( uchar | reserved )
   fragment    =  *( uchar | reserved )

   pchar       = uchar | ":" | "@" | "&" | "="
   uchar       = unreserved | escape
   unreserved  = alpha | digit | safe | extra

   escape      = "%" hex hex
   hex         = digit | "A" | "B" | "C" | "D" | "E" | "F" |
                         "a" | "b" | "c" | "d" | "e" | "f"

   alpha       = lowalpha | hialpha
   lowalpha    = "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" |
                 "j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" |
                 "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z"
   hialpha     = "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" |
                 "J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" |
                 "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z"

   digit       = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" |
                 "8" | "9"

   safe        = "$" | "-" | "_" | "." | "+"
   extra       = "!" | "*" | "'" | "(" | ")" | ","
   national    = "{" | "}" | "|" | "\" | "^" | "~" | "[" | "]" | "`"
   reserved    = ";" | "/" | "?" | ":" | "@" | "&" | "="
   punctuation = "<" | ">" | "#" | "%" | <">
```

2.3. Specific Schemes and their Syntactic Categories

   Each URL scheme has its own rules regarding the presence or absence
   of the syntactic components described in Sections 2.1 and 2.2. In
   addition, some schemes are never appropriate for use with relative
   URLs. However, since relative URLs will only be used within contexts
   in which they are useful, these scheme-specific differences can be
   ignored by the resolution process.

   Within this section, we include as examples only those schemes that
   have a defined URL syntax in RFC 1738 [2]. The following schemes are
   never used with relative URLs:

      mailto   Electronic Mail
      news   USENET news
      telnet   TELNET Protocol for Interactive Sessions

   Some URL schemes allow the use of reserved characters for purposes
   outside the generic-RL syntax given above. However, such use is
   rare. Relative URLs can be used with these schemes whenever the
   applicable base URL follows the generic-RL syntax.

      gopher   Gopher and Gopher+ Protocols
      prospero   Prospero Directory Service
      wais   Wide Area Information Servers Protocol

   Users of gopher URLs should note that gopher-type information is
   almost always included at the beginning of what would be the
   generic-RL path. If present, this type information prevents
   relative-path references to documents with differing gopher-types.

   Finally, the following schemes can always be parsed using the
   generic-RL syntax. This does not necessarily imply that relative
   URLs will be useful with these schemes -- that decision is left to
   the system implementation and the author of the base document.

      file   Host-specific Files
      ftp   File Transfer Protocol
      http   Hypertext Transfer Protocol
      nntp   USENET news using NNTP access

   NOTE: Section 5 of RFC 1738 specifies that the question-mark
         character ("?") is allowed in an ftp or file path segment.
         However, this is not true in practice and is believed to be an
         error in the RFC. Similarly, RFC 1738 allows the reserved
         character semicolon (";") within an http path segment, but does
         not define its semantics; the correct semantics are as defined
         by this document for <params>.

We recommend that new schemes be designed to be parsable via the
generic-RL syntax if they are intended to be used with relative URLs.
A description of the allowed relative forms should be included when a
new scheme is registered, as per Section 4 of RFC 1738 [2].

2.4. Parsing a URL

An accepted method for parsing URLs is useful to clarify the
generic-RL syntax of Section 2.2 and to describe the algorithm for
resolving relative URLs presented in Section 4. This section
describes the parsing rules for breaking down a URL (relative or
absolute) into the component parts described in Section 2.1. The
rules assume that the URL has already been separated from any
surrounding text and copied to a "parse string". The rules are
listed in the order in which they would be applied by the parser.

2.4.1. Parsing the Fragment Identifier

If the parse string contains a crosshatch "#" character, then the
substring after the first (left-most) crosshatch "#" and up to the
end of the parse string is the <fragment> identifier. If the
crosshatch is the last character, or no crosshatch is present, then
the fragment identifier is empty. The matched substring, including
the crosshatch character, is removed from the parse string before
continuing.

Note that the fragment identifier is not considered part of the URL.
However, since it is often attached to the URL, parsers must be able
to recognize and set aside fragment identifiers as part of the
process.

2.4.2. Parsing the Scheme

If the parse string contains a colon ":" after the first character
and before any characters not allowed as part of a scheme name (i.e.,
any not an alphanumeric, plus "+", period ".", or hyphen "-"), the
<scheme> of the URL is the substring of characters up to but not
including the first colon. These characters and the colon are then
removed from the parse string before continuing.

2.4.3. Parsing the Network Location/Login

If the parse string begins with a double-slash "//", then the
substring of characters after the double-slash and up to, but not
including, the next slash "/" character is the network location/login
(<net_loc>) of the URL. If no trailing slash "/" is present, the
entire remaining parse string is assigned to <net_loc>. The double-
slash and <net_loc> are removed from the parse string before

continuing.

2.4.4. Parsing the Query Information

   If the parse string contains a question mark "?" character, then the
   substring after the first (left-most) question mark "?" and up to the
   end of the parse string is the <query> information. If the question
   mark is the last character, or no question mark is present, then the
   query information is empty. The matched substring, including the
   question mark character, is removed from the parse string before
   continuing.

2.4.5. Parsing the Parameters

   If the parse string contains a semicolon ";" character, then the
   substring after the first (left-most) semicolon ";" and up to the end
   of the parse string is the parameters (<params>). If the semicolon
   is the last character, or no semicolon is present, then <params> is
   empty. The matched substring, including the semicolon character, is
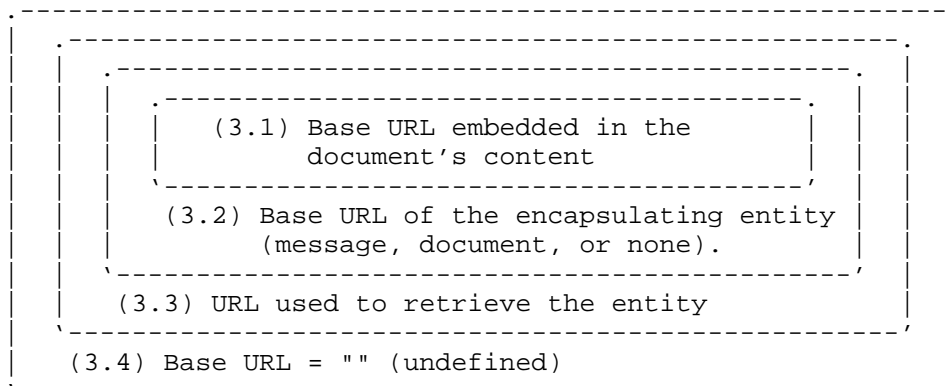   removed from the parse string before continuing.

2.4.6. Parsing the Path

   After the above steps, all that is left of the parse string is the
   URL <path> and the slash "/" that may precede it. Even though the
   initial slash is not part of the URL path, the parser must remember
   whether or not it was present so that later processes can
   differentiate between relative and absolute paths. Often this is
   done by simply storing the preceding slash along with the path.

3. Establishing a Base URL

   The term "relative URL" implies that there exists some absolute "base
   URL" against which the relative reference is applied. Indeed, the
   base URL is necessary to define the semantics of any embedded
   relative URLs; without it, a relative reference is meaningless. In
   order for relative URLs to be usable within a document, the base URL
   of that document must be known to the parser.

The base URL of a document can be established in one of four ways,
listed below in order of precedence. The order of precedence can be
thought of in terms of layers, where the innermost defined base URL
has the highest precedence. This can be visualized graphically as:

```
.-------------------------------------------------------------.
|  .---------------------------------------------------------. |
|  |  .---------------------------------------------------. | |
|  |  |  .-----------------------------------------------. | | |
|  |  |  |       (3.1) Base URL embedded in the          | | | |
|  |  |  |             document's content                | | | |
|  |  |  `-----------------------------------------------' | | |
|  |  |     (3.2) Base URL of the encapsulating entity      | | |
|  |  |           (message, document, or none).             | | |
|  |  `-----------------------------------------------------' | |
|  |       (3.3) URL used to retrieve the entity              | |
|  `---------------------------------------------------------' |
|       (3.4) Base URL = "" (undefined)                        |
`-------------------------------------------------------------'
```

3.1. Base URL within Document Content

   Within certain document media types, the base URL of the document can
   be embedded within the content itself such that it can be readily
   obtained by a parser. This can be useful for descriptive documents,
   such as tables of content, which may be transmitted to others through
   protocols other than their usual retrieval context (e.g., E-Mail or
   USENET news).

   It is beyond the scope of this document to specify how, for each
   media type, the base URL can be embedded. It is assumed that user
   agents manipulating such media types will be able to obtain the
   appropriate syntax from that media type's specification. An example
   of how the base URL can be embedded in the Hypertext Markup Language
   (HTML) [3] is provided in an Appendix (Section 10).

   Messages are considered to be composite documents. The base URL of a
   message can be specified within the message headers (or equivalent
   tagged metainformation) of the message. For protocols that make use
   of message headers like those described in RFC 822 [5], we recommend
   that the format of this header be:

      base-header = "Base" ":" "<URL:" absoluteURL ">"

   where "Base" is case-insensitive and any whitespace (including that
   used for line folding) inside the angle brackets is ignored. For
   example, the header field

Base: <URL:http://www.ics.uci.edu/Test/a/b/c>

would indicate that the base URL for that message is the string
"http://www.ics.uci.edu/Test/a/b/c". The base URL for a message
serves as both the base for any relative URLs within the message
headers and the default base URL for documents enclosed within the
message, as described in the next section.

Protocols which do not use the RFC 822 message header syntax, but
which do allow some form of tagged metainformation to be included
within messages, may define their own syntax for defining the base
URL as part of a message.

3.2. Base URL from the Encapsulating Entity

If no base URL is embedded, the base URL of a document is defined by
the document's retrieval context. For a document that is enclosed
within another entity (such as a message or another document), the
retrieval context is that entity; thus, the default base URL of the
document is the base URL of the entity in which the document is
encapsulated.

Composite media types, such as the "multipart/*" and "message/*"
media types defined by MIME (RFC 1521, [4]), define a hierarchy of
retrieval context for their enclosed documents. In other words, the
retrieval context of a component part is the base URL of the
composite entity of which it is a part. Thus, a composite entity can
redefine the retrieval context of its component parts via the
inclusion of a base-header, and this redefinition applies recursively
for a hierarchy of composite parts. Note that this might not change
the base URL of the components, since each component may include an
embedded base URL or base-header that takes precedence over the
retrieval context.

3.3. Base URL from the Retrieval URL

If no base URL is embedded and the document is not encapsulated
within some other entity (e.g., the top level of a composite entity),
then, if a URL was used to retrieve the base document, that URL shall
be considered the base URL. Note that if the retrieval was the
result of a redirected request, the last URL used (i.e., that which
resulted in the actual retrieval of the document) is the base URL.

3.4. Default Base URL

If none of the conditions described in Sections 3.1 -- 3.3 apply,
then the base URL is considered to be the empty string and all
embedded URLs within that document are assumed to be absolute URLs.

It is the responsibility of the distributor(s) of a document
containing relative URLs to ensure that the base URL for that
document can be established. It must be emphasized that relative
URLs cannot be used reliably in situations where the document's base
URL is not well-defined.

4. Resolving Relative URLs

This section describes an example algorithm for resolving URLs within
a context in which the URLs may be relative, such that the result is
always a URL in absolute form. Although this algorithm cannot
guarantee that the resulting URL will equal that intended by the
original author, it does guarantee that any valid URL (relative or
absolute) can be consistently transformed to an absolute form given a
valid base URL.

The following steps are performed in order:

Step 1: The base URL is established according to the rules of
        Section 3. If the base URL is the empty string (unknown),
        the embedded URL is interpreted as an absolute URL and
        we are done.

Step 2: Both the base and embedded URLs are parsed into their
        component parts as described in Section 2.4.

        a) If the embedded URL is entirely empty, it inherits the
           entire base URL (i.e., is set equal to the base URL)
           and we are done.

        b) If the embedded URL starts with a scheme name, it is
           interpreted as an absolute URL and we are done.

        c) Otherwise, the embedded URL inherits the scheme of
           the base URL.

Step 3: If the embedded URL's <net_loc> is non-empty, we skip to
        Step 7. Otherwise, the embedded URL inherits the <net_loc>
        (if any) of the base URL.

Step 4: If the embedded URL path is preceded by a slash "/", the
        path is not relative and we skip to Step 7.

Step 5: If the embedded URL path is empty (and not preceded by a
        slash), then the embedded URL inherits the base URL path,
        and

        a) if the embedded URL's <params> is non-empty, we skip to
           step 7; otherwise, it inherits the <params> of the base
           URL (if any) and

        b) if the embedded URL's <query> is non-empty, we skip to
           step 7; otherwise, it inherits the <query> of the base
           URL (if any) and we skip to step 7.

Step 6: The last segment of the base URL's path (anything
        following the rightmost slash "/", or the entire path if no
        slash is present) is removed and the embedded URL's path is
        appended in its place. The following operations are
        then applied, in order, to the new path:

        a) All occurrences of "./", where "." is a complete path
           segment, are removed.

        b) If the path ends with "." as a complete path segment,
           that "." is removed.

        c) All occurrences of "<segment>/../", where <segment> is a
           complete path segment not equal to "..", are removed.
           Removal of these path segments is performed iteratively,
           removing the leftmost matching pattern on each iteration,
           until no matching pattern remains.

        d) If the path ends with "<segment>/..", where <segment> is a
           complete path segment not equal to "..", that
           "<segment>/.." is removed.

Step 7: The resulting URL components, including any inherited from
        the base URL, are recombined to give the absolute form of
        the embedded URL.

Parameters, regardless of their purpose, do not form a part of the
URL path and thus do not affect the resolving of relative paths. In
particular, the presence or absence of the ";type=d" parameter on an
ftp URL does not affect the interpretation of paths relative to that
URL. Fragment identifiers are only inherited from the base URL when
the entire embedded URL is empty.

The above algorithm is intended to provide an example by which the
output of implementations can be tested -- implementation of the
algorithm itself is not required. For example, some systems may find
it more efficient to implement Step 6 as a pair of segment stacks
being merged, rather than as a series of string pattern matches.

5. Examples and Recommended Practice

Within an object with a well-defined base URL of

    Base: <URL:http://a/b/c/d;p?q#f>

the relative URLs would be resolved as follows:

5.1. Normal Examples

    g:h   = <URL:g:h>
    g   = <URL:http://a/b/c/g>
    ./g   = <URL:http://a/b/c/g>
    g/   = <URL:http://a/b/c/g/>
    /g   = <URL:http://a/g>
    //g   = <URL:http://g>
    ?y   = <URL:http://a/b/c/d;p?y>
    g?y   = <URL:http://a/b/c/g?y>
    g?y/./x   = <URL:http://a/b/c/g?y/./x>
    #s   = <URL:http://a/b/c/d;p?q#s>
    g#s   = <URL:http://a/b/c/g#s>
    g#s/./x   = <URL:http://a/b/c/g#s/./x>
    g?y#s   = <URL:http://a/b/c/g?y#s>
    ;x   = <URL:http://a/b/c/d;x>
    g;x   = <URL:http://a/b/c/g;x>
    g;x?y#s   = <URL:http://a/b/c/g;x?y#s>
    .   = <URL:http://a/b/c/>
    ./   = <URL:http://a/b/c/>
    ..   = <URL:http://a/b/>
    ../   = <URL:http://a/b/>
    ../g   = <URL:http://a/b/g>
    ../..   = <URL:http://a/>
    ../../   = <URL:http://a/>
    ../../g   = <URL:http://a/g>

5.2. Abnormal Examples

   Although the following abnormal examples are unlikely to occur in
   normal practice, all URL parsers should be capable of resolving them
   consistently. Each example uses the same base as above.

An empty reference resolves to the complete base URL:

    <>   = <URL:http://a/b/c/d;p?q#f>

Parsers must be careful in handling the case where there are more
relative path ".." segments than there are hierarchical levels in the
base URL's path. Note that the ".." syntax cannot be used to change
the <net_loc> of a URL.

    ../../../g   = <URL:http://a/../g>
    ../../../../g = <URL:http://a/../../g>

Similarly, parsers must avoid treating "." and ".." as special when
they are not complete components of a relative path.

    /./g   = <URL:http://a/./g>
    /../g   = <URL:http://a/../g>
    g.   = <URL:http://a/b/c/g.>
    .g   = <URL:http://a/b/c/.g>
    g..   = <URL:http://a/b/c/g..>
    ..g   = <URL:http://a/b/c/..g>

Less likely are cases where the relative URL uses unnecessary or
nonsensical forms of the "." and ".." complete path segments.

    ./../g   = <URL:http://a/b/g>
    ./g/.   = <URL:http://a/b/c/g/>
    g/./h   = <URL:http://a/b/c/g/h>
    g/../h   = <URL:http://a/b/c/h>

Finally, some older parsers allow the scheme name to be present in a
relative URL if it is the same as the base URL scheme. This is
considered to be a loophole in prior specifications of partial URLs
[1] and should be avoided by future parsers.

    http:g   = <URL:http:g>
    http: = <URL:http:>

5.3. Recommended Practice

   Authors should be aware that path names which contain a colon ":"
   character cannot be used as the first component of a relative URL
   path (e.g., "this:that") because they will likely be mistaken for a
   scheme name. It is therefore necessary to precede such cases with
   other components (e.g., "./this:that"), or to escape the colon
   character (e.g., "this%3Athat"), in order for them to be correctly
   parsed. The former solution is preferred because it does not affect
   the absolute form of the URL.

There is an ambiguity in the semantics for the ftp URL scheme
regarding the use of a trailing slash ("/") character and/or a
parameter ";type=d" to indicate a resource that is an ftp directory.
If the result of retrieving that directory includes embedded relative
URLs, it is necessary that the base URL path for that result include
a trailing slash. For this reason, we recommend that the ";type=d"
parameter value not be used within contexts that allow relative URLs.

6. Security Considerations

There are no security considerations in the use or parsing of
relative URLs. However, once a relative URL has been resolved to its
absolute form, the same security considerations apply as those
described in RFC 1738 [2].

7. Acknowledgements

This work is derived from concepts introduced by Tim Berners-Lee and
the World-Wide Web global information initiative. Relative URLs are
described as "Partial URLs" in RFC 1630 [1]. That description was
expanded for inclusion as an appendix for an early draft of RFC 1738,
"Uniform Resource Locators (URL)" [2]. However, after further
discussion, the URI-WG decided to specify Relative URLs separately
from the primary URL draft.

This document is intended to fulfill the recommendations for Internet
Resource Locators as stated in [6]. It has benefited greatly from
the comments of all those participating in the URI-WG. Particular
thanks go to Larry Masinter, Michael A. Dolan, Guido van Rossum, Dave
Kristol, David Robinson, and Brad Barber for identifying
problems/deficiencies in earlier drafts.

8. References

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

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

   [3] Berners-Lee T., and D. Connolly, "HyperText Markup Language
       Specification -- 2.0", Work in Progress, MIT, HaL Computer
       Systems, February 1995.
       <URL:http://www.ics.uci.edu/pub/ietf/html/>

   [4] Borenstein, N., and N. Freed, "MIME (Multipurpose Internet Mail
       Extensions): Mechanisms for Specifying and Describing the Format
       of Internet Message Bodies", RFC 1521, Bellcore, Innosoft,
       September 1993.

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

   [6] Kunze, J., "Functional Recommendations for Internet Resource
       Locators", RFC 1736, IS&T, UC Berkeley, February 1995.

9. Author's Address

Roy T. Fielding
Department of Information and Computer Science
University of California
Irvine, CA 92717-3425
U.S.A.

Tel: +1 (714) 824-4049
Fax: +1 (714) 824-4056
EMail: fielding@ics.uci.edu

10. Appendix - Embedding the Base URL in HTML documents

It is useful to consider an example of how the base URL of a document
can be embedded within the document's content. In this appendix, we
describe how documents written in the Hypertext Markup Language
(HTML) [3] can include an embedded base URL. This appendix does not
form a part of the relative URL specification and should not be
considered as anything more than a descriptive example.

HTML defines a special element "BASE" which, when present in the
"HEAD" portion of a document, signals that the parser should use the
BASE element's "HREF" attribute as the base URL for resolving any
relative URLs. The "HREF" attribute must be an absolute URL. Note
that, in HTML, element and attribute names are case-insensitive. For
example:

```
<!doctype html public "-//IETF//DTD HTML//EN">
<HTML><HEAD>
<TITLE>An example HTML document</TITLE>
<BASE href="http://www.ics.uci.edu/Test/a/b/c">
</HEAD><BODY>
... <A href="../x">a hypertext anchor</A> ...
</BODY></HTML>
```

A parser reading the example document should interpret the given
relative URL "../x" as representing the absolute URL

        <URL:http://www.ics.uci.edu/Test/a/x>

regardless of the context in which the example document was obtained.