Abstract

This document contains a precise specification of how browsers process URLs. The behavior specified in this document might or might not match any particular browser, but browsers might be well-served by adopting the behavior defined herein.
Editorial Note (To be removed by RFC Editor)

If you have suggestions for improving this document, please send email to <mailto:public-iri@w3.org>. Further Working Group information is available from <https://tools.ietf.org/wg/iri/>.

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on October 25, 2011.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the BSD License.
# Table of Contents

1. Open Issues .................................. 4  
2. Definitions ................................... 5  
3. Parsing a URL .................................. 6  
   3.1. Finding the scheme ......................... 6  
   3.2. Finding the authority, path, query, and fragment ...... 7  
   3.3. Finding the user-info, host, and port ............... 8  
   3.4. Find the user name and password .................. 8  
4. Resolving a string relative to a base URL ............... 9  
   4.1. Resolving a string as a relative URL ............... 9  
   4.2. Resolving a string as a scheme-relative URL .......... 10  
   4.3. Resolving a string as an authority-relative URL ....... 11  
   4.4. Resolving a string as a path-relative URL .......... 11  
   4.5. Resolving a string as a query-relative URL .......... 11  
   4.6. Resolving a string as a fragment-relative URL ...... 12  
5. Canonicalizing a URL .......................... 13  
   5.1. Canonicalizing a Scheme ..................... 14  
   5.2. Canonicalizing a User-Info .................... 14  
   5.3. Canonicalizing a Host ....................... 15  
   5.3.1. Host Escape Normalization ................... 15  
   5.4. Canonicalizing a Path ....................... 16  
   5.5. Canonicalizing a Query ....................... 16  
   5.6. Canonicalizing a Fragment .................... 17  
Appendix A. Acknowledgements ........................ 18  
Author’s Address ................................. 19
1. Open Issues

Browsers parse URLs differently depending on which operating system they're running on. The problem is that they want to do sensible things for file paths, but file paths look different on Windows and Unix systems.

How should we handle cases where browsers disagree with the regular expression in RFC 3986? Currently, this document aims to describe how browsers behave, but we'll likely need to compare that to RFC 3986 at some point. Some specific differences that have been brought up on the mailing list:

- http:///example.com/
- http://example.com/
2. Definitions

A control character is a character whose value is less than or equal to U+0020 (" ").

A slash character is either U+002F ("/") or U+005C ("\"). TODO: There’s some question as to whether this is necessary for non-file URLs.

An authority terminating character is either a slash character, U+003F ("?") or U+0023 ("#"). TODO: Why is ";" on this list?

During a parsing algorithm, the remaining string is the characters of the input that have not yet been consumed.
3. Parsing a URL

Given a string of characters, consume all leading and trailing control characters.

Find the scheme, as described in Section ??.

If the algorithm for finding the scheme determines that the URL is invalid:
   -> Abort these steps.

If the scheme is a single upper or lower case ASCII character (TODO: Just ALPHA?):
   -> TODO: Windows drive specs!

If the scheme is a ASCII case-insensitive match for "file":
   -> TODO: File URLs!

If the scheme is a ASCII case-insensitive match for "mailto":
   -> TODO: I think mailto URLs are special, but more testing is required.

If the scheme is hierarchical:
   -> In the after-scheme, if any, find the authority, path, query, and fragment, as described in Section ??.
   -> In the authority, if any, find the user-info, host, and port, as described in Section ??.
   -> In the user-info, if any, find the user name and password, as described in Section ??.
   -> Abort these steps.

The remaining string is the path. TODO: This might not be the best approach. We need to do more testing of data and javascript URLs.

3.1. Finding the scheme

If the remaining string does not contain a ":" character:
   -> The URL is invalid.
Abort these steps.

Consume characters up to, but not including, the first ":" character. These characters are the scheme.

Consume the ":" character.

The remaining characters are the after-scheme.

3.2. Finding the authority, path, query, and fragment

Consume any number of slash characters.

If the remaining string does not contain any authority terminating characters:

- The remaining string is the authority.

- Abort these steps.

Consume characters up to, but not including, the first authority terminating character. The consumed characters are the authority.

If the remaining string does not contain a "?" character or a "#" character:

- The remaining string is the path.

- Abort these steps.

Consume characters up to, but not including, the first "?" or "#" character. The consumed characters are the path.

If the first character of the remaining string is a "?" character:

- Consume the "?" character.

- If the remaining string does not contain a "#" character:

  - The remaining string is the query.

  - Abort these steps.

  - Consume characters up to, but not including, the first "#" character. The consumed characters are the query.

Consume the "#" character.
The remaining string is the fragment.

3.3. Finding the user-info, host, and port

If the remaining string contains an "@" character:

  -> Consume characters up to, but not including the *last* "@" character. The consumed characters are the user-info.

  -> Consume the "@" character.

If the remaining string does not contain an "@" character:

  -> The remaining string is the host.

  -> Abort these steps.

If the first character of the remaining string is a "[" character, the remaining string contains a "]" character, and the last ":" character in the remaining string occurs before the last "]" character in the remaining string:

  -> The remaining string is the host.

  -> Abort these steps.

Consume characters up to, but not including, the last ":" character. The consumed characters are the host.

Consume the ":" character.

The remaining string is the port.

3.4. Find the user name and password

If the remaining string does not contain a ":" character:

  -> The remaining string is the user name.

  -> Abort these steps.

Consume characters up to, but not including, the first ":" character. The consumed characters are the user name.

Consume the ":" character.

The remaining string is the password.
4. Resolving a string relative to a base URL

Given a string relative-url and a ParsedURL base-url, find the scheme of relative-url.

TODO: We probably need to trim leading and trailing control characters.

If relative-url is an invalid URL:

- The resolved URL is relative-url resolved as relative URL.
- Abort these steps.

If relative-url’s scheme contains any characters which are not "valid scheme characters" (TODO: Define valid scheme characters):

- The resolved URL is relative-url resolved as relative URL.
- Abort these steps.

If base-url’s scheme is an ASCII case insensitive match for relative-url’s scheme and the shared scheme is hierarchical:

- The resolved URL is relative-url’s after-scheme resolved as a relative URL.
- Abort these steps.

The resolved URL is relative-url parsed as an absolute URL.

4.1. Resolving a string as a relative URL

Given a string relative-url and a ParsedURL base-url, determine the resolved URL as follows:

TODO: If base-url’s scheme is not hierarchical, we can’t resolve as a relative URL. We’ll probably want to return an invalid URL. Check what happens when resolving an empty string as a relative URL with a non-hierarchical base.

If relative-url is empty:

- The resolved URL is identical to base-url, with the fragment, if any, removed.
- Abort these steps.
If the first character of relative-url is a slash character:

- If relative-url has at least two characters and the second character is also a slash character:
  - The resolved URL is relative-url resolved as a scheme-relative URL.

Otherwise:

- The resolved URL is relative-url resolved as an authority-relative URL.
- Abort these steps.

If the first character of relative-url is a "?" character:

- The resolved URL is relative-url resolved as a query-relative URL.
- Abort these steps.

If the first character of relative-url is a "#" character:

- The resolved URL is relative-url resolved as a fragment-relative URL.
- Abort these steps.

TODO: Think about the case where the relative-url is empty.

The resolved URL is relative-url resolved as a path-relative URL.

4.2. Resolving a string as a scheme-relative URL

Given a string relative-url and a ParsedURL base-url, let resolved-url be

- base-url’s scheme
- concatenated with ":",
- concatenated with relative-url.

The resolved URL is resolved-url parsed as an absolute URL.
4.3. Resolving a string as an authority-relative URL

Given a string relative-url and a ParsedURL base-url, let resolved-url be

- base-url’s scheme
- concatenated with "://",
- concatenated with base-url’s authority,
- concatenated with relative-url.

The resolved URL is resolved-url parsed as an absolute URL.

4.4. Resolving a string as a path-relative URL

TODO: Can the first character of relative-url be a slash character at this point?

TODO: Can we assume base-url is canonicalized here so that it always has at least one "/" character?

Let the directory-name be the characters of the base-url’s path up to and including the last slash character.

Let resolved-url be

- base-url’s scheme
- concatenated with "://",
- concatenated with base-url’s authority,
- concatenated with directory-name.
- concatenated with relative-url.

The resolved URL is resolved-url parsed as an absolute URL.

4.5. Resolving a string as a query-relative URL

Given a string relative-url and a ParsedURL base-url, let resolved-url be

- base-url’s scheme
o concatenated with "://",

o concatenated with base-url’s authority,

o concatenated with base-url’s path,

o concatenated with relative-url.

The resolved URL is resolved-url parsed as an absolute URL.

### 4.6. Resolving a string as a fragment-relative URL

Given a string relative-url and a ParsedURL base-url, let resolved-url be

o base-url’s scheme

o concatenated with "://",

o concatenated with base-url’s authority,

o concatenated with base-url’s path,

o concatenated with “?”,

o concatenated with base-url’s query,

o concatenated with relative-url.

The resolved URL is resolved-url parsed as an absolute URL.
5. Canonicalizing a URL

This section describes how to construct a canonical version of a parsed URL string. TODO: We probably should mention somewhere that there is *not* a unique canonicalization for every URL.

Given parsed URL original-url, if original-url is invalid:

-> Abort these steps.

TODO: Handle file URLs.

If the scheme is hierarchical:

Output the canonicalized scheme (as described in Section ??).

Output "://".

If the user-info is non-empty:

Output the canonicalized user-info (as described in Section ??).

Output ":".

Output the canonicalized host (as described in Section ??).

Let the canonicalized-port be the canonicalized port (as described in Section ??).

If the canonicalized-port is non-empty and is not the default port for the scheme:

Output ":".

Output the canonicalized-port.

Output the canonicalized path (as described in Section ??).

Let the canonicalized-query be the canonicalized query (as described in Section ??).

If the canonicalized-query is non-empty (TODO: Distinguish between empty and non-existent queries):

Output "?".
Output the canonicalized-query.

Let the canonicalized-fragment be the canonicalized fragment (as described in Section ??).

If the canonicalized-fragment is non-empty (TODO: Distinguish between empty and non-existent fragments):

Output "#".

Output the canonicalized-fragment.

5.1. Canonicalizing a Scheme

If the first character of the scheme is not in ALPHA, the scheme is invalid.

Process each character of the scheme in sequence:

If the current character is among ALPHA, DIGIT, "+", "-", and ".":

→ Output the current character.

Otherwise, if the current character is ":"

→ The scheme is invalid.

→ Output the current character.

Otherwise:

→ The scheme is invalid.

→ Output the utf8-percent-escaping of the current character.

5.2. Canonicalizing a User-Info

Process each character of the username in sequence:

If the current character is among TODO:

→ Output the current character.

Otherwise:

→ Output the utf8-percent-escaping of the current character.

If there is no password or if the password is empty:
Abort these steps.

Output ":".

Process each character of the password in sequence:

If the current character is among TODO:

-> Output the current character.

Otherwise:

-> Output the utf8-percent-escaping of the current character.

5.3. Canonicalizing a Host

TODO: Handle IP addresses.

Let unicode-host be the host-escape-normalized host (see Section ??).

Output result of applying the IDNA to-ascii algorithm to the unicode-host. TODO: Properly reference IDNA’s to-ascii algorith (we might need a wrapper like we do in the cookie spec).

5.3.1. Host Escape Normalization

\[
\text{host-escaped} = \text{U+0000-U+002A} / \text{U+002C} / \text{U+002F} / \text{U+003B-U+0040} / \text{U+005C} / \\
\text{U+005E} / \text{U+0060} / \text{U+007B-U+007F}
\]

Process each character of the host in sequence:

If the current character is ":%":

-> TODO: Handle percent-unescaping.

If the current character matches host-escaped:

-> Output the utf8-percent-escaping of the current character.

Otherwise, if the current character matches ALPHA:

-> Output the current character converted to lower case.

Otherwise:

-> Output the current character.
5.4. Canonicalizing a Path

TODO: Do we need to ensure that path’s always start with a slash character?

If the path is empty:

  -> Output "/" and abort these steps.

Path-escaped = U+0000-U+0020 / U+0022-U+0023 / U+0025 / U+003C / U+003E /
             U+003F / U+005C / U+005E / U+0060 / U+007B-U+007D / U+007F
Path-unescaped = "-" / DIGIT / ALPHA / "_" / "~"

Process each character of the path in sequence:

  If the current character matches path-escaped or is greater than
  or equal to U+0080:

    -> Output the utf8-percent-escaping of the current character.

  Otherwise, if the current character is ".":

    -> TODO: Handle "." collapsing.

  Otherwise, if the current character is ":":

    -> Output "/".

  Otherwise, if the current character is "%":

    -> TODO: Handle percent-unescaping.

  Otherwise:

    -> Output the current character.

5.5. Canonicalizing a Query

TODO: Handle the ambient encoding case.

Process each character of the query in sequence:

  If the current character is among TODO:

    -> Output the current character.

  Otherwise:
5.6. Canonicalizing a Fragment

Process each character of the fragment in sequence:

If the current character has a Unicode value greater than or equal to U+0020:

  -> Output the current character.

Otherwise:

  -> Output the utf8-percent-escaping of the current character.

Note: The above algorithm results in the canonicalized fragment containing non-US-ASCII characters.
Appendix A. Acknowledgements

TODO