NAME

uri, url, urn − uniform resource identifier (URI), including a URL or URN

SYNOPSIS

URI = [absoluteURI | relativeURI] ["#" fragment]

absoluteURI = scheme ":" (hierarchical_part | opaque_part)

relativeURI = (net_path | absolute_path | relative_path) ["?" query]

scheme = "http" | "ftp" | "gopher" | "mailto" | "news" | "telnet" |
"file" | "man" | "info" | "whatis" | "ldap" | "wais" | ...

hierarchical_part = (net_path | absolute_path) ["?" query]

net_path = "//" authority [absolute_path]

absolute_path = "/" path_segments

relative_path = relative_segment [absolute_path]

DESCRIPTION

A Uniform Resource Identifier (URI) is a short string of characters identifying an abstract or physical resource (for example, a web page). A Uniform Resource Locator (URL) is a URI that identifies a resource through its primary access mechanism (e.g., its network "location"), rather than by name or some other attribute of that resource. A Uniform Resource Name (URN) is a URI that must remain globally unique and persistent even when the resource ceases to exist or becomes unavailable.

URIs are the standard way to name hypertext link destinations for tools such as web browsers. The string "http://www.kernel.org" is a URL (and thus it is also a URI). Many people use the term URL loosely as a synonym for URI (though technically URLs are a subset of URIs).

URIs can be absolute or relative. An absolute identifier refers to a resource independent of context, while a relative identifier refers to a resource by describing the difference from the current context. Within a relative path reference, the complete path segments "." and ".." have special meanings: "the current hierarchy level" and "the level above this hierarchy level", respectively, just like they do in UNIX-like systems. A path segment which contains a colon character can't be used as the first segment of a relative URI path (e.g., "this:that"), because it would be mistaken for a scheme name; precede such segments with ./ (e.g., "./this:that"). Note that descendants of MS-DOS (e.g., Microsoft Windows) replace devicename colons with the vertical bar ("|") in URIs, so "C:" becomes "C|".

A fragment identifier, if included, refers to a particular named portion (fragment) of a resource; text after a '#' identifies the fragment. A URI beginning with '#' refers to that fragment in the current resource.

Usage

There are many different URI schemes, each with specific additional rules and meanings, but they are intentionally made to be as similar as possible. For example, many URL schemes permit the authority to be the following format, called here an *ip_server* (square brackets show what's optional):

ip_server = [*user* [: *password*] @] *host* [: *port*]

This format allows you to optionally insert a username, a user plus password, and/or a port number. The *host* is the name of the host computer, either its name as determined by DNS or an IP address (numbers separated by periods). Thus the URI <http://fred:fredpassword@example.com:8080/> logs into a web server on host example.com as fred (using fredpassword) using port 8080. Avoid including a password in a URI if possible because of the many security risks of having a password written down. If the URL supplies a username but no password, and the remote server requests a password, the program interpreting the URL should request one from the user.

Here are some of the most common schemes in use on UNIX-like systems that are understood by many tools. Note that many tools using URIs also have internal schemes or specialized schemes; see those tools' documentation for information on those schemes.

http − Web (HTTP) server

http://*ip_server*/*path*
http://*ip_server*/*path*?*query*

This is a URL accessing a web (HTTP) server. The default port is 80. If the path refers to a directory, the web server will choose what to return; usually if there is a file named "index.html" or "index.htm" its content is returned, otherwise, a list of the files in the current directory (with appropriate links) is generated and returned. An example is <http://lwn.net>.

A query can be given in the archaic "isindex" format, consisting of a word or phrase and not including an equal sign (=). A query can also be in the longer "GET" format, which has one or more query entries of the form *key=value* separated by the ampersand character (&). Note that *key* can be repeated more than once, though it's up to the web server and its application programs to determine if there's any meaning to that. There is an unfortunate interaction with HTML/XML/SGML and the GET query format; when such URIs with more than one key are embedded in SGML/XML documents (including HTML), the ampersand (&) has to be rewritten as &. Note that not all queries use this format; larger forms may be too long to store as a URI, so they use a different interaction mechanism (called POST) which does not include the data in the URI. See the Common Gateway Interface specification at ⟨http://www.w3.org/CGI⟩ for more information.

ftp − File Transfer Protocol (FTP)

ftp://*ip_server*/*path*

This is a URL accessing a file through the file transfer protocol (FTP). The default port (for control) is 21. If no username is included, the username "anonymous" is supplied, and in that case many clients provide as the password the requestor's Internet email address. An example is <ftp://ftp.is.co.za/rfc/rfc1808.txt>.

gopher − Gopher server

gopher://*ip_server*/*gophertype selector*
gopher://*ip_server*/*gophertype selector*%09*search*
gopher://*ip_server*/*gophertype selector*%09*search*%09*gopher+_string*

The default gopher port is 70. *gophertype* is a single-character field to denote the Gopher type of the resource to which the URL refers. The entire path may also be empty, in which case the delimiting "/" is also optional and the gophertype defaults to "1".

selector is the Gopher selector string. In the Gopher protocol, Gopher selector strings are a sequence of octets which may contain any octets except 09 hexadecimal (US-ASCII HT or tab), 0A hexadecimal (US-ASCII character LF), and 0D (US-ASCII character CR).

mailto − Email address

mailto:*email-address*

This is an email address, usually of the form *name@hostname*. See**mailaddr**(7) for more information on the correct format of an email address. Note that any % character must be rewritten as %25. An example is <mailto:dwheeler@dwheeler.com>.

news − Newsgroup or News message

news:*newsgroup-name*
news:*message-id*

A *newsgroup-name* is a period-delimited hierarchical name, such as "comp.infosystems.www.misc". If <newsgroup-name> is "*" (as in <news:*>), it is used to refer to "all available news groups". An example is <news:comp.lang.ada>.

A *message-id* corresponds to the Message-ID of IETF RFC 1036, ⟨http://www.ietf.org/rfc/rfc1036.txt⟩ without the enclosing "<" and ">"; it takes the form *unique@ full_domain_name*. A message identifier may be distinguished from a news group name by the presence of the "@" character.

telnet − Telnet login

telnet://*ip_server*/

The Telnet URL scheme is used to designate interactive text services that may be accessed by the Telnet protocol. The final "/" character may be omitted. The default port is 23. An example is <telnet://melvyl.ucop.edu/>.

file − Normal file

file://*ip_server*/*path_segments*
file:*path_segments*

This represents a file or directory accessible locally. As a special case,*ip_server* can be the string "localhost" or the empty string; this is interpreted as "the machine from which the URL is being interpreted". If the path is to a directory, the viewer should display the directory's contents with links to each containee; not all viewers currently do this. KDE supports generated files through the URL <file:/cgi-bin>. If the given file isn't found, browser writers may want to try to expand the filename via filename globbing (see **glob**(7) and **glob**(3)).

The second format (e.g., <file:/etc/passwd>) is a correct format for referring to a local file. However, older standards did not permit this format, and some programs don't recognize this as a URI. A more portable syntax is to use an empty string as the server name, for example, <file:///etc/passwd>; this form does the same thing and is easily recognized by pattern matchers and older programs as a URI. Note that if you really mean to say "start from the current location", don't specify the scheme at all; use a relative address like <../test.txt>, which has the side-effect of being scheme-independent. An example of this scheme is <file:///etc/passwd>.

man − Man page documentation

man:*command-name*
man:*command-name*(*section*)

This refers to local online manual (man) reference pages. The command name can optionally be followed by a parenthesis and section number; see **man**(7) for more information on the meaning of the section numbers. This URI scheme is unique to UNIX-like systems (such as Linux) and is not currently registered by the IETF. An example is <man:ls(1)>.

info − Info page documentation

info:*virtual-filename*
info:*virtual-filename*#*nodename*
info:(*virtual-filename*)
info:(*virtual-filename*)*nodename*

This scheme refers to online info reference pages (generated from texinfo files), a documentation format used by programs such as the GNU tools. This URI scheme is unique to UNIX-like systems (such as Linux) and is not currently registered by the IETF. As of this writing, GNOME and KDE differ in their URI syntax and do not accept the other's syntax. The first two formats are the GNOME format; in nodenames all spaces are written as underscores. The second two formats are the KDE format; spaces in nodenames must be written as spaces, even though this is forbidden by the URI standards. It's hoped that in the future most tools will understand all of these formats and will always accept underscores for spaces in nodenames. In both GNOME and KDE, if the form without the nodename is used the nodename is assumed to be "Top". Examples of the GNOME format are <info:gcc> and <info:gcc#G++_and_GCC>. Examples of the KDE format are <info:(gcc)> and <info:(gcc)G++ and GCC>.

whatis − Documentation search

whatis:*string*

This scheme searches the database of short (one-line) descriptions of commands and returns a list of descriptions containing that string. Only complete word matches are returned. See **whatis**(1). This URI scheme is unique to UNIX-like systems (such as Linux) and is not currently registered by the IETF.

ghelp − GNOME help documentation

ghelp:*name-of-application*

This loads GNOME help for the given application. Note that not much documentation currently exists in this format.

ldap − Lightweight Directory Access Protocol

ldap://*hostport*
ldap://*hostport*/
ldap://*hostport*/*dn*
ldap://*hostport*/*dn*?*attributes*
ldap://*hostport*/*dn*?*attributes*?*scope*
ldap://*hostport*/*dn*?*attributes*?*scope*?*filter*
ldap://*hostport*/*dn*?*attributes*?*scope*?*filter*?*extensions*

This scheme supports queries to the Lightweight Directory Access Protocol (LDAP), a protocol for querying a set of servers for hierarchically organized information (such as people and computing resources). See RFC 2255 ⟨http://www.ietf.org/rfc/rfc2255.txt⟩ for more information on the LDAP URL scheme. The components of this URL are:

hostport
: the LDAP server to query, written as a hostname optionally followed by a colon and the port number. The default LDAP port is TCP port 389. If empty, the client determines which the LDAP server to use.

dn
: the LDAP Distinguished Name, which identifies the base object of the LDAP search (see RFC 2253 ⟨http://www.ietf.org/rfc/rfc2253.txt⟩ section 3).

attributes
: a comma-separated list of attributes to be returned; see RFC 2251 section 4.1.5. If omitted, all attributes should be returned.

scope
: specifies the scope of the search, which can be one of "base" (for a base object search), "one" (for a one-level search), or "sub" (for a subtree search). If scope is omitted, "base" is assumed.

filter
: specifies the search filter (subset of entries to return). If omitted, all entries should be returned. See RFC 2254 ⟨http://www.ietf.org/rfc/rfc2254.txt⟩ section 4.

extensions
: a comma-separated list of type=value pairs, where the =value portion may be omitted for options not requiring it. An extension prefixed with a '!' is critical (must be supported to be valid), otherwise it is noncritical (optional).

LDAP queries are easiest to explain by example. Here's a query that asks ldap.itd.umich.edu for information about the University of Michigan in the U.S.:

ldap://ldap.itd.umich.edu/o=University%20of%20Michigan,c=US

To just get its postal address attribute, request:

ldap://ldap.itd.umich.edu/o=University%20of%20Michigan,c=US?postalAddress

To ask a host.com at port 6666 for information about the person with common name (cn) "Babs Jensen" at University of Michigan, request:

ldap://host.com:6666/o=University%20of%20Michigan,c=US??sub?(cn=Babs%20Jensen)

wais − Wide Area Information Servers

wais://*hostport*/*database*
wais://*hostport*/*database*?*search*
wais://*hostport*/*database*/*wtype*/*wpath*

This scheme designates a WAIS database, search, or document (see IETF RFC 1625 ⟨http://www.ietf.org/rfc/rfc1625.txt⟩ for more information on WAIS). Hostport is the hostname, optionally followed by a colon and port number (the default port number is 210).

The first form designates a WAIS database for searching. The second form designates a particular search of

the WAIS database *database*. The third form designates a particular document within a WAIS database to be retrieved. *wtype* is the WAIS designation of the type of the object and *wpath* is the WAIS document-id.

other schemes

There are many other URI schemes. Most tools that accept URIs support a set of internal URIs (e.g., Mozilla has the about: scheme for internal information, and the GNOME help browser has the toc: scheme for various starting locations). There are many schemes that have been defined but are not as widely used at the current time (e.g., prospero). The nntp: scheme is deprecated in favor of the news: scheme. URNs are to be supported by the urn: scheme, with a hierarchical name space (e.g., urn:ietf:... would identify IETF documents); at this time URNs are not widely implemented. Not all tools support all schemes.

Character encoding

URIs use a limited number of characters so that they can be typed in and used in a variety of situations.

The following characters are reserved, that is, they may appear in a URI but their use is limited to their reserved purpose (conflicting data must be escaped before forming the URI):

```
; / ? : @ & = + $ ,
```

Unreserved characters may be included in a URI. Unreserved characters include uppercase and lowercase Latin letters, decimal digits, and the following limited set of punctuation marks and symbols:

```
- _ . ! ~ * ' ( )
```

All other characters must be escaped. An escaped octet is encoded as a character triplet, consisting of the percent character "%" followed by the two hexadecimal digits representing the octet code (you can use uppercase or lowercase letters for the hexadecimal digits). For example, a blank space must be escaped as "%20", a tab character as "%09", and the "&" as "%26". Because the percent "%" character always has the reserved purpose of being the escape indicator, it must be escaped as "%25". It is common practice to escape space characters as the plus symbol (+) in query text; this practice isn't uniformly defined in the relevant RFCs (which recommend %20 instead) but any tool accepting URIs with query text should be prepared for them. A URI is always shown in its "escaped" form.

Unreserved characters can be escaped without changing the semantics of the URI, but this should not be done unless the URI is being used in a context that does not allow the unescaped character to appear. For example, "%7e" is sometimes used instead of "~" in an HTTP URL path, but the two are equivalent for an HTTP URL.

For URIs which must handle characters outside the US ASCII character set, the HTML 4.01 specification (section B.2) and IETF RFC 3986 (last paragraph of section 2.5) recommend the following approach:

(1) translate the character sequences into UTF-8 (IETF RFC 3629)—see **utf−8**(7)—and then

(2) use the URI escaping mechanism, that is, use the %HH encoding for unsafe octets.

Writing a URI

When written, URIs should be placed inside double quotes (e.g., "http://www.kernel.org"), enclosed in angle brackets (e.g., <http://lwn.net>), or placed on a line by themselves. A warning for those who use double-quotes: **never** move extraneous punctuation (such as the period ending a sentence or the comma in a list) inside a URI, since this will change the value of the URI. Instead, use angle brackets instead, or switch to a quoting system that never includes extraneous characters inside quotation marks. This latter system, called the 'new' or 'logical' quoting system by "Hart's Rules" and the "Oxford Dictionary for Writers and Editors", is preferred practice in Great Britain and in various European languages. Older documents suggested inserting the prefix "URL:" just before the URI, but this form has never caught on.

The URI syntax was designed to be unambiguous. However, as URIs have become commonplace, traditional media (television, radio, newspapers, billboards, etc.) have increasingly used abbreviated URI references consisting of only the authority and path portions of the identified resource (e.g., <www.w3.org/Addressing>). Such references are primarily intended for human interpretation rather than machine, with the assumption that context-based heuristics are sufficient to complete the URI (e.g., hostnames beginning with "www" are likely to have a URI prefix of "http://" and hostnames beginning with "ftp" likely to have a prefix of "ftp://"). Many client implementations heuristically resolve these references. Such heuristics may

change over time, particularly when new schemes are introduced. Since an abbreviated URI has the same syntax as a relative URL path, abbreviated URI references cannot be used where relative URIs are permitted, and can be used only when there is no defined base (such as in dialog boxes). Don't use abbreviated URIs as hypertext links inside a document; use the standard format as described here.

STANDARDS

(IETF RFC 2396) ⟨http://www.ietf.org/rfc/rfc2396.txt⟩, (HTML 4.0) ⟨http://www.w3.org/TR/REC−html40⟩.

NOTES

Any tool accepting URIs (e.g., a web browser) on a Linux system should be able to handle (directly or indirectly) all of the schemes described here, including the man: and info: schemes. Handling them by invoking some other program is fine and in fact encouraged.

Technically the fragment isn't part of the URI.

For information on how to embed URIs (including URLs) in a data format, see documentation on that format. HTML uses the format <A HREF="*uri*"> *text* </A>. Texinfo files use the format @uref{*uri*}. Man and mdoc have the recently added UR macro, or just include the URI in the text (viewers should be able to detect :// as part of a URI).

The GNOME and KDE desktop environments currently vary in the URIs they accept, in particular in their respective help browsers. To list man pages, GNOME uses <toc:man> while KDE uses <man:(index)>, and to list info pages, GNOME uses <toc:info> while KDE uses <info:(dir)> (the author of this man page prefers the KDE approach here, though a more regular format would be even better). In general, KDE uses <file:/cgi-bin/> as a prefix to a set of generated files. KDE prefers documentation in HTML, accessed via the <file:/cgi-bin/helpindex>. GNOME prefers the ghelp scheme to store and find documentation. Neither browser handles file: references to directories at the time of this writing, making it difficult to refer to an entire directory with a browsable URI. As noted above, these environments differ in how they handle the info: scheme, probably the most important variation. It is expected that GNOME and KDE will converge to common URI formats, and a future version of this man page will describe the converged result. Efforts to aid this convergence are encouraged.

Security

A URI does not in itself pose a security threat. There is no general guarantee that a URL, which at one time located a given resource, will continue to do so. Nor is there any guarantee that a URL will not locate a different resource at some later point in time; such a guarantee can be obtained only from the person(s) controlling that namespace and the resource in question.

It is sometimes possible to construct a URL such that an attempt to perform a seemingly harmless operation, such as the retrieval of an entity associated with the resource, will in fact cause a possibly damaging remote operation to occur. The unsafe URL is typically constructed by specifying a port number other than that reserved for the network protocol in question. The client unwittingly contacts a site that is in fact running a different protocol. The content of the URL contains instructions that, when interpreted according to this other protocol, cause an unexpected operation. An example has been the use of a gopher URL to cause an unintended or impersonating message to be sent via a SMTP server.

Caution should be used when using any URL that specifies a port number other than the default for the protocol, especially when it is a number within the reserved space.

Care should be taken when a URI contains escaped delimiters for a given protocol (for example, CR and LF characters for telnet protocols) that these are not unescaped before transmission. This might violate the protocol, but avoids the potential for such characters to be used to simulate an extra operation or parameter in that protocol, which might lead to an unexpected and possibly harmful remote operation to be performed.

It is clearly unwise to use a URI that contains a password which is intended to be secret. In particular, the use of a password within the "userinfo" component of a URI is strongly recommended against except in those rare cases where the "password" parameter is intended to be public.

BUGS

Documentation may be placed in a variety of locations, so there currently isn't a good URI scheme for general online documentation in arbitrary formats. References of the form <file:///usr/doc/ZZZ> don't work because different distributions and local installation requirements may place the files in different directories (it may be in /usr/doc, or /usr/local/doc, or /usr/share, or somewhere else). Also, the directory ZZZ usually changes when a version changes (though filename globbing could partially overcome this). Finally, using the file: scheme doesn't easily support people who dynamically load documentation from the Internet (instead of loading the files onto a local filesystem). A future URI scheme may be added (e.g., "userdoc:") to permit programs to include cross-references to more detailed documentation without having to know the exact location of that documentation. Alternatively, a future version of the filesystem specification may specify file locations sufficiently so that the file: scheme will be able to locate documentation.

Many programs and file formats don't include a way to incorporate or implement links using URIs.

Many programs can't handle all of these different URI formats; there should be a standard mechanism to load an arbitrary URI that automatically detects the users' environment (e.g., text or graphics, desktop environment, local user preferences, and currently executing tools) and invokes the right tool for any URI.

SEE ALSO

lynx(1), **man2html**(1), **mailaddr**(7), **utf−8**(7)

IETF RFC 2255 ⟨http://www.ietf.org/rfc/rfc2255.txt⟩