

UriFrame

Web Uniform Resource Identifier Framework

Version 0.4
30 June 2003

Neil W. Van Dyke
neil@neilvandyke.org

<http://www.neilvandyke.org/uriframe/>

Copyright © 2003 Neil W. Van Dyke. This program is Free Software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version. This program is distributed in the hope that it will be useful, but without any warranty; without even the implied warranty of merchantability or fitness for a particular purpose. See the GNU Lesser General Public License [LGPL] for more details.

1 Introduction

NOTICE: This is an snapshot of work in progress. The documentation for existing code is incomplete, with many "!!!" placeholders. Expect substantial API changes.

UriFrame is a extensible object class framework for parsing, representing, manipulating, and formatting Web Uniform Resource Identifiers (URI) [RFC2396], which includes Uniform Resource Locators (URL). It uses the PLT Scheme object model.

UriFrame has standard support for many popular URI schemes, and contains components that can be used to add support for different schemes. Mixins are used extensively, and class-mapping abstractions support easy addition of application-specific URI behavior. UriFrame is currently oriented towards applications that need to support multiple URI schemes and that often require a full parse of each URI. (Applications that only need to support one URI scheme, such as an HTTP server, are likely better off using an application-specific URI representation.)

Programs using UriFrame should require the `(lib "uriframe.ss" "uriframe")` module, and may also need `(lib "class.ss")`.

2 Basic URI Object Interface

All URI classes within UriFrame implement the `uri<%>` interface.

`uri<%>` [Interface]
Basic interface for URI and URI reference objects. Notably, all URI classes must implement the `parse/layout!` operation, which parses a URI string given generic syntax layout information per [RFC2396].

fragment [Method on uri<%>]

set-fragment! *v* [Method on uri<%>]

This pair of methods gets and sets (respectively) the fragment attribute of a URI reference. The fragment attribute is either a string or **#f**, and is represented unescaped. For example:

```
(send (string->uri "http://a/b#My%20%231%20Foo") fragment)
⇒ "My #1 Foo"
```

uri-scheme [Method on uri<%>]

Yields the scheme ID of the URI, which is a lowercase symbol as generated by **string->uri-scheme**.

parse/layout! *str a o f base* [Method on uri<%>]

Sent by **string->uri** on a freshly-instantiated **uri<%>** object, the **parse/layout!** message should result in the parsing of string *str* with the **regexp** match position pairs *a*, *o*, and *f* (authority, opaque part, fragment). *base* is either **#f** or a **uri<%>** that should be treated as the base URI for resolving relative aspects.

Subclasses implementing this method normally propagate the message to their superclass before performing any other action.

absolute? [Method on uri<%>]

Predicate for whether or not the URI is absolute.

force-absolute! [Method on uri<%>]

Force a URI to be absolute, to the extent possible with the given information, such as by discarding excess relative path components. Note that a URI might remain relative despite this operation, such as when the scheme is unknown or there is some scheme-specific information that cannot be resolved. For example:

```
(send* (string->uri "foo") (force-absolute!) (absolute?)) ⇒ #f
```

resolve! *base* [Method on uri<%>]

Resolves relative components of the URI with respect to base URI *base*, yielding the resolved URI and possibly mutating the receiver.

If the URI scheme of *base* is unknown, then behavior is undefined. However, *base* does not necessarily have to be absolute in all its components (e.g., it may consist of only a scheme, or it may have a relative path), since in practice this may be sensitive to the URI scheme. Implementing classes are free to ignore non-absolute components of *base*.

The usual case is for the receiver of this message to be mutated and returned. However, resolving a URI without an explicit scheme against a base URI with a scheme can result in a new URI object being constructed for the scheme class, so the code pattern **(set! u1 (send u1 resolve! u2))** may be helpful.

resolve/like! *base* [Method on uri<%>]

!!!

->string [Method on uri<%>]
Yields a string representation of the URI in the canonical format, with proper escaping.
!!!

write-uri *port* [Method on uri<%>]
Writes the same representation as `->string` to output port *port*, so the following two expressions are functionally equivalent, although the first may be more efficient.

```
(send u write-uri p)  
≡  
(display (send u ->string) p)
```

!!!

3 Default Base Class

The default base class implementing `uri<%>` is provided by `mixin uri-base-mixin`, which can have `object%` or an application-specific class as a superclass.

uri-base-mixin [Mixin]
Implements interface `uri<%>` as a base class (e.g., does not propagate `parse/layout!` to its superclass). Implements `absolute?`, `resolve!`, and `write-uri` with the help of methods that can be overridden in subclasses. Also provides methods used by procedure `uri->plt-url`.

authority-absolute? [Method on uri-base-mixin]
opaque-absolute? [Method on uri-base-mixin]
!!! override in subclasses

resolve/like! *base* [Method on uri-base-mixin]
Resolve the URI with respect to base URI *base*, which is of the same URI scheme. This method will normally be overridden for each URI scheme class. !!! propagate up
!!!

write-authority-part *port* [Method on uri-base-mixin]
write-opaque-part *port* [Method on uri-base-mixin]
!!! override in subclasses

plt-url-scheme	[Method on uri-base-mixin]
plt-url-host	[Method on uri-base-mixin]
plt-url-portnum	[Method on uri-base-mixin]
plt-url-path	[Method on uri-base-mixin]
plt-url-params	[Method on uri-base-mixin]
plt-url-query	[Method on uri-base-mixin]
plt-url-fragment	[Method on uri-base-mixin]

These methods are used in the construction of PLT urls by the `->plt-url` method of this mixin, and can be overridden in subclasses. `plt-url-portnum` should yield a whole number or `#f`, and each other method should yield a string or `#f`.

4 Escaping

!!! [RFC2396 sec. 2.4]

!!! many applications will not call directly, since uri classes escape and unescape automatically from converting to/from canonical string format

uri-unescape *str* [*start-k* [*end-k* [*always-copy?*]]] [Procedure]

uri-unplusescape *str* [*start-k* [*end-k* [*always-copy?*]]] [Procedure]

!!!

uri-escape *str* [*start-k* [*end-k* [*always-copy?*]]] [Procedure]

uri-plusescape *str* [*start-k* [*end-k* [*always-copy?*]]] [Procedure]

!!! escapes all except alphanum, minus, underscore, period, and tilde.

!!! plusescape discouraged

5 Components

!!!

5.1 Server Authorities

!!! [RFC2396 sec. 3.2]

server-authority-uri<%> [Interface]

!!! [RFC2396 sec. 3.2.2]

server-userinfo [Method on server-authority<%>]

server-host [Method on server-authority<%>]

server-portnum [Method on server-authority<%>]

!!!

default-server-portnum [Method on server-authority<%>]

!!! defined by the scheme class/mixin !!! users don't normal call directly

effective-server-portnum [Method on `server-authority<%>`]
!!! call this to get the port number for initiating connections, etc

server-authority-uri-mixin [Mixin]
!!!

5.2 Hierarchical Paths

!!! segment names representation:

```
(send (string->uri "http://a/b") path-segments) => ("a" "b" )  
(send (string->uri "http://a/b/") path-segments) => ("a" "b" #f)  
(send (string->uri "http://a/b/.") path-segments) => !!!
```

!!! parameters representation:

```
(send (string->uri "http://host/aaa/bbb;p1;p2/ccc;p3/ddd")  
      path-segments)  
=> ("aaa" ("bbb" "p1" "p2") ("ccc" "p3") "ddd")
```

uri-path-segment-name *segment* [Procedure]

uri-path-segment-params *segment* [Procedure]

!!! effect of mutating these values is undefined

hierarchical-uri<%> [Interface]

!!! note that last segment can have empty name

path-segments [Method on `hierarchical-uri<%>`]

set-path-segments! *v* [Method on `hierarchical-uri<%>`]

!!! effect of mutating is undefined

path-uplevels [Method on `hierarchical-uri<%>`]

set-path-uplevels! *v* [Method on `hierarchical-uri<%>`]

!!!

```
(send (string->uri "http://a/b") path-uplevels) => 0  
(send (string->uri "http://a/b/") path-uplevels) => #f  
(send (string->uri "http://./a/b") path-uplevels) => 0  
(send (string->uri "http://./a/b") path-uplevels) => #f  
(send (string->uri "http://./a/b") path-uplevels) => 1  
(send (string->uri "http://././a/b") path-uplevels) => 2  
(send (string->uri "http://./././a/b") path-uplevels) => 3  
(send (string->uri "http://a/./b") path-uplevels) => #f  
(send (string->uri "http://a/././b") path-uplevels) => 1  
(send (string->uri "http://a/./././b") path-uplevels) => 2  
(send (string->uri "http://a/./b") path-uplevels) => 0  
(send (string->uri "http://a/././b") path-uplevels) => 1  
(send (string->uri "http://a/./././b") path-uplevels) => 2
```

path-absolute? [Method on `hierarchical-uri<%>`]
Predicate for whether or not path is absolute.

force-path-absolute! [Method on `hierarchical-uri<%>`]
Forces the path to be absolute. `path-uplevels` is set to `#f`. If there are no segments, the segments are set to one segment with an empty name.

resolve-path! *base* [Method on `hierarchical-uri<%>`]
!!!

path-string [Method on `hierarchical-uri<%>`]
write-path *port* [Method on `hierarchical-uri<%>`]
!!!

hierarchical-uri-mixin [Mixin]
!!!

5.3 HTTP-like Queries

!!!
(send (string->uri "http://h/srch?kw=fiendish+scheme&icase&x=1%2B2")
query-attrs)
⇒ (("kw" . "fiendish scheme") ("icase" . #t) ("x" . "1+2"))

httplike-query-uri<%> [Interface]
!!!

parse-query *str start-k end-k* [Method on `httplike-query-uri<%>`]
!!!

query-attrs [Method on `httplike-query-uri<%>`]
!!!

!!!

query-string [Method on `httplike-query-uri<%>`]
write-query *port* [Method on `httplike-query-uri<%>`]
!!!

httplike-query-uri-mixin [Mixin]
Accepts a `uri<%>` and implements `httplike-query-uri<%>`.

6 URI Scheme Classes

!!! interface and mixin pairs.

!!! interfaces define the uri scheme's interface, mixins supply scheme-specific behavior that is not supplied by the component mixins above.

!!! some mixins have dependencies that can be supplied by the above component mixins or user-supplied mixins or classes.

6.1 file

file-uri<%> [Interface]
!!!

file-uri-mixin [Mixin]
!!!

6.2 ftp

ftp-uri<%> [Interface]
!!!

ftp-uri-mixin [Mixin]
!!!

6.3 gopher

gopher-uri<%> [Interface]
!!!

gopher-uri-mixin [Mixin]
!!!

6.4 http

http-uri<%> [Interface]
!!!

http-uri-mixin [Mixin]
!!!

6.5 https

https-uri<%> [Interface]
!!!

https-uri-mixin [Mixin]
!!!

6.6 imap

!!! [RFC2192]

imap-uri<%> [Interface]
!!!

imap-uri-mixin [Mixin]
!!!

6.7 ipp

ipp-uri<%> [Interface]
!!!

ipp-uri-mixin [Mixin]
!!!

6.8 javascript

javascript-uri<%> [Interface]
!!!

javascript-uri-mixin [Mixin]
!!!

6.9 mailto

!!! [RFC2368]

mailto-uri<%> [Interface]
!!!

mailto-uri-mixin [Mixin]
!!!

6.10 news

`news-uri<%>` [Interface]
!!!

`news-uri-mixin` [Mixin]
!!!

6.11 nfs

!!! [RFC2224]

`nfs-uri<%>` [Interface]
!!!

`nfs-uri-mixin` [Mixin]
!!!

6.12 telnet

`telnet-uri<%>` [Interface]
!!!

`telnet-uri-mixin` [Mixin]
!!!

6.13 urn

!!!

`urn-uri<%>` [Interface]
!!!

`urn-namespace` [Method on `urn-uri<%>`]
!!!

`urn-uri-mixin` [Mixin]
!!!

7 Mapping URI Schemes to Classes

!!! this is used for adding new uri schemes, changing what uri schemes are supported, adding a new uri base type, or changing individual uri class mappings. users who just want the default uri scheme classes and mapping can ignore this section and just use `string->uri` and `uri-scheme->class` procedures.

7.1 URI Scheme Identifiers

string->uri-scheme *str* [Procedure]

Accepts a URI scheme name in string form and yields a scheme identifier, which is a lowercase symbol:

(string->uri-scheme "HtTp") ⇒ http

7.2 Unknown Scheme Class

!!! why unknown scheme class is needed (for representing unknown scheme uri, especially til can be resolved), and how it differs (resolving unknown-scheme object can instantiate a new object of different class)

!!! Per [RFC2396 sec. 3.1], relative URI do not specify a scheme, although in practice they can have a scheme. Oracle example.

unknown-scheme-uri<%> [Interface]

!!!

set-uri-scheme! *v* [Method on unknown-scheme-uri<%>]

!!! note doesn't cause a reparse/remap

unknown-scheme-uri-mixin [Mixin]

!!! note that parsing *does* copy raw-str. this is because we're keeping the string around for possible resolution later, and might be parsing from a reusable string buffer that's written in blocks, for example.

7.3 Mapper Procedure

!!!

make-uri-scheme-mapper *alist* [*unknown*] [Procedure]

!!!

make-standard-uri-scheme-mapper [*base* [*unknown?* [*mix*]]] [Procedure]

!!!

uri-scheme-mapper [Parameter]
!!!

uri-scheme->class *uri-scheme* [Procedure]
Using the `uri-scheme-mapper` parameter, yields either a `uri<%>` class for *uri-scheme*, or `#f` if there is no class for that scheme.
(implementation? (uri-scheme->class 'ftp) uri<%>) ⇒ #t

8 String Conversion

Two of the most commonly used procedures in application code will be those that convert URI objects to and from the standard string syntax: `string->uri` and `uri->string`.

string/layout/scheme/base->uri *str a o f uri-scheme base-uri* [Procedure]
This method is used by the `string->uri` procedure, and also by the `resolve!` method of `unknown-scheme-uri-mixin` objects.

string->uri *str [base-uri [start-k [end-k]]]* [Procedure]
The usual interface for constructing `uri<%>` objects. Accepts a relative or absolute URI or URI reference in string form *str* and yields either a `uri<%>` object or `#f` (if the URI scheme is unknown and no `unknown-scheme` class is used). The `uri-scheme-mapper` parameter is used for scheme class lookup.

If the optional *base-uri* (which defaults to `#f`) is non-`#f`, it is a `uri<%>` to use as the base URI for resolving relative URI resulting from the parsing of *str*. Although a relative URI can be resolved with respect to a base URI after construction, supplying the base as the *base-uri* term here is generally more efficient.

The optional *start-k* and *end-k* can be used to restrict parsing of *str* to the given substring, with `#f` for either term meaning the start or end of *str*, respectively.

uri->string *uri* [Procedure]
Yields `uri<%>` *uri* in canonical string form. This is equivalent to sending the `->string` message to *uri*.
!!!

9 PLT URL Conversion

!!! note about loading (lib "uriframe-plt-url.ss" "uriframe")

!!! issue with parsing and representing parameters

```
(define s "http://h/foo;p1/bar;p2;p3")  
(url-params (string->url s)) ⇒ "p1/bar;p2;p3"  
(url-params (uri->plt-url (string->uri s))) ⇒ "p1;p2;p3"
```

uri->plt-url *uri* [Procedure]
!!!

`plt-url->uri url`

[Procedure]

!!! uses `uri-scheme-mapper` parameter

History

Version 0.4 — 30 June 2003

Snapshot with latest versions of Httpper and SRFI-19.

Version 0.3 — 1 May 2003

Paths like `/` and `foo/.` are now parsed more consistently. PLT URL conversion code broken off into separate module, to save memory (approx. 900 KB on author's machine) when not needed.

Version 0.2 — 3 April 2003

Another snapshot, including a small test suite.

Version 0.1 — 30 March 2003

Snapshot of a work in progress, to solicit comments on the API.

References

[LGPL] Free Software Foundation, "GNU Lesser General Public License," Version 2.1, February 1999, 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.

<http://www.gnu.org/copyleft/lesser.html>

[RFC2192]

C. Newman, "IMAP URL Scheme," IETF RFC 2192, September 1997.

<http://www.ietf.org/rfc/rfc2192.txt>

[RFC2224]

B. Callaghan, "NFS URL Scheme," IETF RFC 2224, October 1997.

<http://www.ietf.org/rfc/rfc2224.txt>

[RFC2368]

P. Hoffman, L. Masinter, J. Zawinski, "The mailto URL scheme," IETF RFC 2368, July 1998.

<http://www.ietf.org/rfc/rfc2368.txt>

[RFC2396]

T. Berners-Lee, R. Fielding, L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax," IETF RFC 2396, August 1998.

<http://www.ietf.org/rfc/rfc2396.txt>

[RFC2732]

R. Hinden, B. Carpenter, L. Masinter, "Format for Literal IPv6 Addresses in URL's," IETF RFC 2732, December 1999.

<http://www.ietf.org/rfc/rfc2732.txt>

[RFC3305]

M. Mealling, R. Denenberg, (eds.), "Report from the Joint W3C/IETF URI Planning Interest Group: Uniform Resource Identifiers (URIs), URLs, and Uniform Resource Names (URNs): Clarifications and Recommendations," IETF RFC 3305, August 2002.

<http://www.ietf.org/rfc/rfc3305.txt>