

uri.scm

Web Uniform Resource Identifiers (URI) in Scheme

Version 0.1
18 August 2004

Neil W. Van Dyke
neil@neilvandyke.org

<http://www.neilvandyke.org/uri-scm/>

Copyright © 2004 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 details.

1 Introduction

Note: This version of the library has endured some testing, and is being used in at least one production application, but be advised that some design refinements and API changes are expected. Especially, we are reconsidering the use of immutable strings and pairs, would like to add more of the extensibility of [UriFrame], and need to look at the forthcoming IETF-W3C standards.

`uri.scm` is a Scheme code library for parsing, representing, and transforming Web Uniform Resource Identifiers (URI) [RFC2396], which includes Uniform Resource Locators (URL) and Uniform Resource Names (URN). It supports absolute and relative URIs and URI references.

The library provides two separate interfaces, based on the two supported representations: a convenient verbatim string representation, and a parsed representation. From an interface standpoint, URI are immutable objects, and all operations are functions yielding the same or new immutable URI objects. Functionality specific to individual URI schemes is generally outside the scope of this library, and is better supported via separate companion libraries.

This library has been designed after experience with [UriFrame], which was specific to PLT Scheme and dependent on a heavyweight object system. This library appears in some ways similar to the `uri` module of [SLIB], but is intended to provide additional functionality.

The current version of this library is specific to PLT Scheme, but it has been written with the intention of being portable shortly to most R5RS Scheme implementations that support popular SRFIs. It officially requires [SRFI-6], [SRFI-8], [SRFI-9], [SRFI-13] (for

`string-downcase`), [SRFI-16], [SRFI-23], and [SRFI-39].¹ It also requires some regular expression operations that can be provided by the [Pregexp] library if the implementation does not provide appropriate native operations. If the implementation provides immutable strings and pairs, the library will take advantage of them; on implementations that do not provide these, copying and hopeful wishes will be used.

2 Escaping and Unescaping

Several procedures to support escaping and unescaping of URI component strings, as described in [RFC2396 sec. 2.4], are provided. Also provided are escaping and unescaping procedures that also support `+` as an encoding of a space character, as is used in some HTTP encodings of HTML forms.

These procedures have multiple variants, concerning mutability of the strings they yield, and following the naming convention:

foo Always yields a new, mutable string.

foo-i Always yields an immutable string (or a new string, if the Scheme implementation does not support immutable string).

foo/shareok
If the output is equal to the input, might yield the input string rather than yielding a copy of it.

Many applications will not call these procedures directly, since most of this library's interface automatically escapes and unescapes strings as appropriate.

uri-escape *str* [*start* [*end*]] \Rightarrow *string* [Procedure]

uri-escape-i *str* [*start* [*end*]] \Rightarrow *string* [Procedure]

uri-escape/shareok *str* [*start* [*end*]] \Rightarrow *string* [Procedure]

Yields a URI-escaped encoding of string *str*. If *start* and *end* are given, then they designate the substring of *str* to use. All characters are escaped, except alphanumerics, minus, underscore, period, and tilde. For example.

`(uri-escape "a = b/c + d") \Rightarrow "a%20%3D%20b%2Fc%20%2B%20d"`

uri-plusescape *str* [*start* [*end*]] \Rightarrow *string* [Procedure]

uri-plusescape-i *str* [*start* [*end*]] \Rightarrow *string* [Procedure]

uri-plusescape/shareok *str* [*start* [*end*]] \Rightarrow *string* [Procedure]

Like `uri-escape`, except encodes space characters as `+` instead of `%20`. This should generally only be used to mimic the encoding some Web browsers do of HTML form values. For example:

`(uri-plusescape "a = b/c + d") \Rightarrow "a+%3D+b%2Fc+%2B+d"`

¹ Scheme implementors are encouraged to support SRFI-0 and SRFI-7, and to define one or more SRFI-0 features that identify their language variant and implementation. The lukewarm popularity of these two SRFIs is a barrier to maintaining portable versions of this library.

uri-unescape *str* [*start* [*end*]] ⇒ *string* [Procedure]

uri-unescape-i *str* [*start* [*end*]] ⇒ *string* [Procedure]

uri-unescape/shareok *str* [*start* [*end*]] ⇒ *string* [Procedure]

Yields an URI-unescaped string from the encoding in string *str*. If *start* and *end* are given, then they designate the substring of *str* to use. For example:

```
(uri-unescape "a%20b+c%20d") ⇒ "a b+c d"
```

uri-unplusescape *str* [*start* [*end*]] ⇒ *string* [Procedure]

uri-unplusescape-i *str* [*start* [*end*]] ⇒ *string* [Procedure]

uri-unplusescape/shareok *str* [*start* [*end*]] ⇒ *string* [Procedure]

Like `uri-unescape`, but also decodes the plus (+) character as to space character. For example:

```
(uri-unplusescape "a%20b+c%20d") ⇒ "a b c d"
```

char->uri-escaped-string *chr* ⇒ *string* [Procedure]

char->uri-escaped-string-i *chr* ⇒ *string* [Procedure]

Yields a URI-escaped of character *chr*. For example:

```
(char->uri-escaped-string #\) ⇒ "%2F"
```

3 String URI API

This section describes the “URI string” API, while the next section describes the “URI object,” (`uriobj`) API. All procedures in this section yield URIs using immutable strings, and accept URIs as strings (immutable or mutable) or as the opaque objects described in the next section.

3.1 Writing URIs to Ports and Converting URIs to Strings

display-uri *uri port* ⇒ *undef* [Procedure]

display-uri/nofragment *uri port* ⇒ *undef* [Procedure]

Displays *uri* to output port *port*. For example:

```
(display-uri "http://s/foo#bar" (current-output-port))
+ http://s/foo#bar
(display-uri/nofragment "http://s/foo#bar" (current-output-port))
+ http://s/foo
```

uri->string *uri* ⇒ *string* [Procedure]

Yields the full string representation of URI *uri*. Of course this is not needed when using only the string representation of URI, but using this procedure in libraries permits the `uriobj` to also be used. For example:

```
(define my-uriobj (string->uriobj "http://www/"))
my-uriobj           ⇒ #<uriobj>
(uri->string my-uriobj) ⇒ "http://www/"
(uri->string "http://www/") ⇒ "http://www/"
```

3.2 URI Schemes

URI schemes are currently represented as lowercase Scheme symbols and associated data.

ftp-uri-scheme \Rightarrow *urischeme* [Variable]
gopher-uri-scheme \Rightarrow *urischeme* [Variable]
http-uri-scheme \Rightarrow *urischeme* [Variable]
https-uri-scheme \Rightarrow *urischeme* [Variable]
imap-uri-scheme \Rightarrow *urischeme* [Variable]
ipp-uri-scheme \Rightarrow *urischeme* [Variable]
news-uri-scheme \Rightarrow *urischeme* [Variable]
nfs-uri-scheme \Rightarrow *urischeme* [Variable]
telnet-uri-scheme \Rightarrow *urischeme* [Variable]

Some common URI scheme symbols, as a convenience for Scheme code that must be portable to Scheme implementations with case-insensitive readers. For example, in some Scheme implementations:

```
'ftp           $\Rightarrow$  FTP  
ftp-uri-scheme  $\Rightarrow$  ftp
```

uri-scheme *uri* \Rightarrow *urischeme* [Procedure]

Yields the URI scheme of *uri*, or #f if none can be determined. For example:

```
(uri-scheme "Http://www")  $\Rightarrow$  http
```

register-uri-scheme-default-portnum *sym portnum* \Rightarrow *undef* [Procedure]

Registers integer *portnum* as the default port number for the server authority component of URI scheme *sym*.

```
(define x-foo-uri-scheme (string->symbol "x-foo"))  
(register-uri-scheme-default-portnum x-foo-uri-scheme 007)  
(register-uri-scheme-default-portnum x-foo-uri-scheme 666)  
error cannot change uri scheme default portnum: x-foo 7 666
```

register-uri-scheme-hierarchical *sym* \Rightarrow *undef* [Procedure]

Registers URI scheme *sym* as having a “hierarchical” form as described in [RFC2396 sec. 3].

3.3 URI Reference Fragment Identifiers

uri-fragment *uri* \Rightarrow *string-or-f* [Procedure]

uri-fragment/escaped *uri* \Rightarrow *string-or-f* [Procedure]

Yields the fragment identifier component of URI (or URI reference) *uri* as a string, or #f if there is no fragment. **uri-fragment** yields the fragment in unescaped form, and **uri-fragment/escaped** yields an escaped form in the unusual case that is desired. For example:

```
(uri-fragment      "foo#a%20b")  $\Rightarrow$  "a b"  
(uri-fragment/escaped "foo#a%20b")  $\Rightarrow$  "a%20b"
```

uri-without-fragment *uri* ⇒ *string* [Procedure]

Yields *uri* without the fragment component. For example:

```
(uri-without-fragment "http://w/#bar") ⇒ "http://w/"
```

uri-with-fragment *uri fragment* ⇒ *string* [Procedure]

uri-with-fragment/escaped *uri fragment* ⇒ *string* [Procedure]

Yields a URI that is like *uri* except with the fragment *fragment* (or no fragment if *fragment* is #f). For example:

```
(uri-with-fragment "http://w/" "foo") ⇒ "http://w/#foo"
```

```
(uri-with-fragment "http://w/#foo" "bar") ⇒ "http://w/#bar"
```

```
(uri-with-fragment "http://w/#bar" #f) ⇒ "http://w/"
```

The `uri-with-fragment/escaped` variant can be used when the desired fragment string is already in URI-escaped form:

```
(uri-with-fragment "foo" "a b") ⇒ "foo#a%20b"
```

```
(uri-with-fragment/escaped "foo" "a%20b") ⇒ "foo#a%20b"
```

3.4 Hierarchical URIs

This and some of the following subsections concern “hierarchical” generic URI syntax as described in [RFC2396 sec. 3].

uri-hierarchical? *uri* ⇒ *boolean* [Procedure]

Yields a Boolean value for whether or not the URI scheme of URI *uri* is known to have a “hierarchical” generic URI layout. For example:

```
(uri-hierarchical? "http://www/") ⇒ #t
```

```
(uri-hierarchical? "mailto://www/") ⇒ #f
```

```
(uri-hierarchical? "//www/") ⇒ #f
```

3.5 Server-Based Naming Authorities

Several procedures extract the server authority values from URIs [RFC2396 sec. 3.2.2].

uri-server-userinfo+host+portnum *uri* ⇒ (*string-or-f*, *string-or-f*, *integer-or-f*) [Procedure]

Yields three values for the server authority of URI *uri*: the userinfo as a string (or #f), the host as a string (or #f), and the effective port number as an integer (or #f). The effective port number of a server authority defaults to the default of the URI scheme unless overridden. For example (note the effective port number is 21, the default for the `ftp` scheme):

```
(uri-server-userinfo+host+portnum "ftp://anon@ftp.foo.bar/")
```

```
⇒ "anon" "ftp.foo.bar" 21
```

uri-server-userinfo *uri* ⇒ *string-of-f* [Procedure]

uri-server-host *uri* ⇒ *string-of-f* [Procedure]

uri-server-portnum *uri* ⇒ *integer-or-f* [Procedure]

Yield the respective part of the server authority of *uri*. See the discussion of `uri-server-userinfo+host+portnum`.

3.6 Hierarchical Paths

A parsed hierarchical path [RFC2396 sec. 3] is represented in `uri.scm` as a tuple of a list of path segments and an *upcount*. The list of path segments does not contain any “.” or “..” relative components, as those are removed during parsing. The upcount is either `#f`, meaning an absolute path, or an integer 0 or greater, meaning a relative path of that many levels “up.” A path segment without any parameters is represented as either a string or, if empty, `#f`. For example:

```
(uri-path-upcount+segments "/a/b/")           ⇒ #f ("a" "b" #f)
(uri-path-upcount+segments "/a/b/c")         ⇒ #f ("a" "b" "c")
(uri-path-upcount+segments "/a/../../b/c")    ⇒ 2  ("b" "c")
```

and:

```
(uri-path-upcount+segments "/.") ⇒ #f ()
(uri-path-upcount+segments "/")  ⇒ #f (#f)
(uri-path-upcount+segments ".")  ⇒ 0  ()
(uri-path-upcount+segments "")    ⇒ 0  (#f)
(uri-path-upcount+segments "./")  ⇒ 0  (#f)
(uri-path-upcount+segments "..")  ⇒ 1  ()
(uri-path-upcount+segments "../") ⇒ 1  ()
(uri-path-upcount+segments "../../") ⇒ 1  (#f)
```

A path segment with parameters is represented as a list, with the first element a string or `#f` for the path name, and the remaining elements strings for the parameters. For example:

```
(uri-path-segments "../../a/b;p1/c/d;p2;p3;p4")
⇒ ("a" ("b" "p1") "c" ("d" "p2" "p3") (#f "p4"))
```

In the current version of `uri.scm`, parsed paths are actually represented in reverse, which simplifies path resolution and permits list tails to be shared among potentially large numbers of long paths. For example (`uripath` is a concept of the “object URI” API):

```
(let ((base (string->uripath "/a/b/c/index.html")))
  (map (lambda (n)
        (resolved-uripath (string->uripath n) base))
       '("x.html" "y/y.html" "../../z/z.html")))
⇒
(("x.html" . #0=("c" . #1=("b" "a")))
 ("y.html" "y" . #0#)
 ("z.html" "z" . #1#))
```

uri-path-upcount+segments *uri* ⇒ (*integer-or-f*, *list-of-urisegment*) [Procedure]

uri-path-upcount+segments/reverse *uri* ⇒ (*integer-or-f*, *list-of-urisegment*) [Procedure]

Yields the path upcount and the segments of *uri* as two values. The segments list should be considered immutable, as it might be shared elsewhere. `uri-path-upcount+segments/reverse` yields the segments list in reverse order, and is the more efficient of the two procedures.

```
(uri-path-upcount+segments/reverse "../../a/../../b/./c")
```

```

⇒ 2 ("c" "b")
(uri-path-upcount+segments    "../a/../../b/./c")
⇒ 2 ("b" "c")

```

uri-path-upcount *uri* ⇒ *integer-or-f* [Procedure]

uri-path-segments *uri* ⇒ *list-of-urisegment* [Procedure]

uri-path-segments/reverse *uri* ⇒ *list-of-urisegment* [Procedure]

See the documentation for `uri-path-upcount+segments`.

```

(uri-path-upcount    "../a/../../b/./c") ⇒ 2
(uri-path-segments  "../a/../../b/./c") ⇒ ("b" "c")
(uri-path-segments/reverse "../a/../../b/./c") ⇒ ("c" "b")

```

urisegment-name *urisegment* ⇒ *string-or-f* [Procedure]

urisegment-params *urisegment* ⇒ *list* [Procedure]

urisegment-name+params *urisegment* ⇒ (*string-or-f*, *list*) [Procedure]

urisegment-has-params? *urisegment* ⇒ *boolean* [Procedure]

Yield the components of a parsed URI segment. The values should be considered immutable. For example:

```

(urisegment-name+params "foo")           ⇒ "foo" ()
(urisegment-name+params #f)             ⇒ #f   ()
(urisegment-name+params '("foo" "p1" "p2")) ⇒ "foo" ("p1" "p2")
(urisegment-name+params '#f "p1" "p2")) ⇒ #f   ("p1" "p2")

```

3.7 Attribute-Value Queries

This library provides support for parsing the URI query component [RFC2396 sec. 3.4], as attribute-value lists in the manner of `http` URI scheme queries. Parsed queries are represented as association lists, in which the *car* of each pair is the attribute name as a string, and the *cdr* is either the attribute value as a string or `#t` if no value given. All strings are URI-unesaped. For example:

```

(uri-query "?q=fiendish+scheme&case&x=&y=1%2B2")
⇒
(("q" . "fiendish scheme") ("case" . #t) ("x" . "") ("y" . "1+2"))

```

uri-query *uri* ⇒ *uriquery* [Procedure]

Yields the parsed attribute-value query of *uri*, or `#f` if no query. For example:

```

(uri-query "?x=42&y=1%2B2") ⇒ (("x" . "42") ("y" . "1+2"))

```

uri-query-value *uri attr* ⇒ *string-or-t-or-f* [Procedure]

Yields the value of attribute *attr* in *uri*'s query, or `#f` if *uri* has no query component or no *attr* attribute. If the attribute appears multiple times in the query, the value of the first occurrence is used. For example:

```

(uri-query-value "?x=42&y=1%2B2" "y") ⇒ "1+2"

```

uriquery-value *uriquery attr* ⇒ *string-or-t-or-f* [Procedure]
Yields the value of attribute *attr* in *uriquery*, or **#f** if there is no such attribute. If the attribute appears multiple times in the query, the value of the first occurrence is used.

3.8 Resolving Relative URI

This subsection concerns resolving relative URI.

absolute-uri? *uri* ⇒ *boolean* [Procedure]
Yields a Boolean value for whether or not URI *uri* is *known* by the library's criteria to be absolute.

resolved-uri *uri base-uri* ⇒ *string* [Procedure]
Yields a URI string that is URI *uri* possibly resolved with respect to URI *base-uri*, but not necessarily absolute. As an extension to [RFC2396] rules for resolution, *base-uri* may be a relative URI.

```
(resolved-uri "x.html" "http://w/a/b/c.html")  
⇒ "http://w/a/b/x.html"  
(resolved-uri "//www:80/" "http:")  
⇒ "http://www/"
```

absolute-uri *uri* ⇒ *string* [Procedure]
Yields a URI that may be a variation on *uri* that has been forced to absolute (by, e.g., dropping relative path components, or supplying a missing path). The result might not be an absolute URI, however, due to limitations of the library or insufficient information in the URI. For example:

```
(absolute-uri "http://w/./a") ⇒ "http://w/a"  
(absolute-uri "http://w") ⇒ "http://w/"
```

normalized-uri *uri* ⇒ *string* [Procedure]
Yields a possibly “normalized” variation on URI *uri*, such as by consistent use of escaping. The exact behavior of this procedure will change in future versions of the library.

4 Object URI API

Note: The Object URI API is only sparsely documented, although many of its procedures have analogues in the String URI API, which is documented in the preceding section.

4.1 Predicate

uriobj? *v* [Procedure]

4.2 Converting Strings to URI Objects

string->uriobj <i>str</i> ⇒ <i>uriobj</i>	[Procedure]
string/base->uriobj <i>str base-uri</i> ⇒ <i>uriobj</i>	[Procedure]
string/base-uriobj->uriobj <i>str base-uriobj</i> ⇒ <i>uriobj</i>	[Procedure]
substring->uriobj <i>str start end</i> ⇒ <i>uriobj</i>	[Procedure]
substring/base->uriobj <i>str start end base-uri</i> ⇒ <i>uriobj</i>	[Procedure]
substring/base-uriobj->uriobj <i>str start end base-uriobj</i> ⇒ <i>uriobj</i>	[Procedure]
uri->uriobj <i>uri</i> ⇒ <i>uriobj</i>	[Procedure]

4.3 Writing URIs to Ports and Converting URIs to Strings

display-uriobj <i>uriobj port</i> ⇒ <i>undef</i>	[Procedure]
display-uriobj/nofragment <i>uriobj port</i> ⇒ <i>undef</i>	[Procedure]
uriobj->string <i>uriobj</i> ⇒ <i>string</i>	[Procedure]
uriobj->string/nofragment <i>uriobj</i> ⇒ <i>string</i>	[Procedure]

4.4 URI Schemes

uriobj-scheme <i>uriobj</i> ⇒ <i>urischeme</i>	[Procedure]
uriobj-with-scheme <i>uriobj urischeme</i> ⇒ <i>uriobj</i>	[Procedure]
string->urischeme <i>str</i> ⇒ <i>urischeme</i>	[Procedure]
symbol->urischeme <i>sym</i> ⇒ <i>urischeme</i>	[Procedure]
urischeme->string ⇒ <i>string</i>	[Procedure]
urischeme-hierarchical? <i>urischeme</i>	[Procedure]
urischeme-default-portnum <i>urischeme</i> ⇒ <i>integer-or-f</i>	[Procedure]

4.5 URI Reference Fragments

uriobj-fragment <i>uriobj</i> ⇒ <i>string-or-f</i>	[Procedure]
uriobj-fragment/escaped <i>uriobj</i> ⇒ <i>string-or-f</i>	[Procedure]
uriobj-with-fragment <i>uriobj fragment</i> ⇒ <i>uriobj</i>	[Procedure]
uriobj-with-fragment/escaped <i>uriobj fragment</i> ⇒ <i>uriobj</i>	[Procedure]

4.6 Hierarchical URIs

uriobj-hierarchical? <i>uriobj</i> ⇒ <i>boolean</i>	[Procedure]
--	-------------

4.7 Server Authorities

uriobj-uriserver <i>uriobj</i> ⇒ <i>uriserver</i>	[Procedure]
uriobj-uriserver+path+query <i>uri</i> ⇒ (<i>uriserver</i> , <i>uripath</i> , <i>uriquery</i>)	[Procedure]
uri-uriserver <i>uri</i> ⇒ <i>uriserver</i>	[Procedure]
uri-uriserver+uripath+uriquery <i>uri</i> ⇒ (<i>uriserver</i> , <i>uripath</i> , <i>uriquery</i>)	[Procedure]
uriobj-userinfo+host+portnum <i>uriobj</i> ⇒ (<i>string-or-f</i> , <i>string-or-f</i> , <i>integer-or-f</i>)	[Procedure]
uriobj-portnum <i>uriobj</i> ⇒ <i>integer-or-f</i>	[Procedure]
make-uriserver <i>userinfo host portnum</i> ⇒ <i>uriserver</i>	[Procedure]
make-uriserver/default-portnum <i>userinfo host portnum</i> <i>default-portnum</i> ⇒ <i>uriserver</i>	[Procedure]
make-or-reuse-uriserver <i>userinfo host portnum base-uriserver</i> ⇒ <i>uriserver</i>	[Procedure]
make-or-reuse-uriserver/default-portnum <i>userinfo host portnum</i> <i>base-uriserver default-portnum</i> ⇒ <i>uriserver</i>	[Procedure]
string->uriserver <i>str</i> ⇒ <i>uriserver</i>	[Procedure]
string/base->uriserver <i>str base-uriserver</i> ⇒ <i>uriserver</i>	[Procedure]
string/default-portnum->uriserver <i>str default-portnum</i> ⇒ <i>uriserver</i>	[Procedure]
string/base/default-portnum->uriserver <i>str base-uriserver</i> <i>default-portnum</i> ⇒ <i>uriserver</i>	[Procedure]
substring->uriserver <i>str start end</i> ⇒ <i>uriserver</i>	[Procedure]
substring/base->uriserver <i>str start end base-uriserver</i> ⇒ <i>uriserver</i>	[Procedure]
substring/default-portnum->uriserver <i>str start end</i> <i>default-portnum</i> ⇒ <i>uriserver</i>	[Procedure]
substring/base/default-portnum->uriserver <i>str start end</i> <i>base-uriserver default-portnum</i> ⇒ <i>uriserver</i>	[Procedure]
uriserver-userinfo <i>uriserver</i> ⇒ <i>string-or-f</i>	[Procedure]
uriserver-host <i>uriserver</i> ⇒ <i>string-or-f</i>	[Procedure]
uriserver-portnum <i>uriserver</i> ⇒ <i>integer-or-f</i>	[Procedure]
uriserver-userinfo+host+portnum <i>uriserver</i> ⇒ (<i>string-or-f</i> , <i>string-or-f</i> , <i>integer-or-f</i>)	[Procedure]
write-uriserver <i>uriserver port</i>	[Procedure]
uriserver-with-default-portnum <i>uriserver default-portnum</i> ⇒ <i>uriserver</i>	[Procedure]
resolved-uriserver <i>uriserver base-uriserver</i> ⇒ <i>uriserver</i>	[Procedure]
resolved-uriserver/default-portnum <i>uriserver base-uriserver</i> <i>default-portnum</i> ⇒ <i>uriserver</i>	[Procedure]

4.8 Hierarchical Paths

uri-path <i>uri</i> ⇒ <i>uripath-or-f</i>	[Procedure]
uri-path/noparams <i>uri</i> ⇒ <i>uripath-or-f</i>	[Procedure]
uriobj-uri-path <i>uriobj</i> ⇒ <i>uripath-or-f</i>	[Procedure]
uriobj-uri-path/noparams <i>uriobj</i> ⇒ <i>uripath-or-f</i>	[Procedure]
make-uri-path <i>upcount segments</i> ⇒ <i>uripath</i>	[Procedure]
make-uri-path/reverse <i>upcount segments</i> ⇒ <i>uripath</i>	[Procedure]
make-uri-path/reverse/shareok <i>upcount segments</i> ⇒ <i>uripath</i>	[Procedure]
uripath-with-upcount <i>uripath upcount</i> ⇒ <i>uripath</i>	[Procedure]
string->uripath <i>str</i> ⇒ <i>uripath</i>	[Procedure]
string/base->uripath <i>str base-uripath</i> ⇒ <i>uripath</i>	[Procedure]
substring->uripath <i>str start end</i> ⇒ <i>uripath</i>	[Procedure]
substring/base->uripath <i>str start end base-uripath</i> ⇒ <i>uripath</i>	[Procedure]
uripath-upcount <i>uripath</i> ⇒ <i>integer-or-f</i>	[Procedure]
uripath-segments <i>uripath</i> ⇒ <i>list</i>	[Procedure]
uripath-segments/reverse <i>uripath</i> ⇒ <i>list</i>	[Procedure]
uripath-upcount+segments <i>uripath</i> ⇒ (<i>integer-or-f, list</i>)	[Procedure]
uripath-upcount+segments/reverse <i>uripath</i> ⇒ (<i>integer-or-f, list</i>)	[Procedure]
uripath-has-params? <i>uripath</i> ⇒ <i>boolean</i>	[Procedure]
write-uri-path <i>uripath port</i> ⇒ <i>undef</i>	[Procedure]
write-uri-path/leading-slash <i>uripath port</i> ⇒ <i>undef</i>	[Procedure]
uripath->string <i>uripath</i> ⇒ <i>string</i>	[Procedure]
uripath->string/leading-slash <i>uripath</i> ⇒ <i>string</i>	[Procedure]
(uri-path-segments "//a/b") ⇒ ("b")	
(uri-path-segments "../a/b") ⇒ (#f "a" "b")	
(uripath->string (string->uripath "//b"))	
⇒ "//b"	
(uripath->string/leading-slash (string->uripath "//b"))	
⇒ "../b"	
(uripath->string/leading-slash (string->uripath "/a/b"))	
⇒ "/a/b"	
(uripath->string/leading-slash (string->uripath "/;p1/b"))	
⇒ "/;p1/b"	
resolved-uri-path <i>uripath base-uripath</i> ⇒ <i>uripath</i>	[Procedure]
absolute-uri-path <i>uripath</i> ⇒ <i>uripath</i>	[Procedure]

4.9 Attribute-Value Queries

uriobj-uriquery <i>uriobj</i> ⇒ <i>uriquery</i>	[Procedure]
--	-------------

string->uriquery *str* [Procedure]
substring->uriquery *str start end* [Procedure]
write-uriquery *uriquery port* ⇒ *undef* [Procedure]

4.10 Resolving Relative URI

absolute-uriobj? *uriobj* ⇒ *boolean* [Procedure]
resolved-uriobj *uriobj base-uri* ⇒ *uriobj* [Procedure]
resolved-uriobj/base-uriobj *uriobj base-uriobj* ⇒ *uriobj* [Procedure]
absolute-uriobj *uriobj* ⇒ *uriobj* [Procedure]

5 Tests

The `uri.scm` source code file defines a regression test suite for the library itself, in procedure `uri-internal:test`. This test suite can be disabled in the source code.

History

Version 0.1 — 18-Aug-2004
Initial release. Incorporates some code from UriFrame.

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>
- [Pregexp] Dorai Sitaram, “pregexp: Portable Regular Expressions for Scheme and Common Lisp,” version 1e9.
<http://www.ccs.neu.edu/home/dorai/pregexp/pregexp.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>
- [SLIB] Aubrey Jaffer, "SLIB," version 3a1, 30 November 2003.
<http://swiss.csail.mit.edu/~jaffer/SLIB>
- [SRFI-6] William D. Clinger, "Basic String Ports," SRFI 6, 1 July 1999.
<http://srfi.schemers.org/srfi-6/srfi-6.html>
- [SRFI-8] John David Stone, "receive: Binding to multiple values," 30-Aug-1999.
<http://srfi.schemers.org/srfi-8/srfi-8.html>
- [SRFI-9] Richard Kelsey, "Defining Record Types," SRFI 9, 9 September 1999.
<http://srfi.schemers.org/srfi-9/srfi-9.html>
- [SRFI-13] Olin Shivers, "String Libraries," SRFI 13, 23 December 2000.
<http://srfi.schemers.org/srfi-13/srfi-13.html>
- [SRFI-16] Lars T Hansen, "Syntax for procedures of variable arity," SRFI 16, 10 March 2000.
<http://srfi.schemers.org/srfi-16/srfi-16.html>
- [SRFI-23] Stephan Houben, "Error reporting mechanism," SRFI 23, 26 April 2001.
<http://srfi.schemers.org/srfi-23/srfi-23.html>
- [SRFI-39] Marc Feeley, "Parameter objects," SRFI 39, 30 June 2003.
<http://srfi.schemers.org/srfi-39/srfi-39.html>
- [UriFrame] Neil W. Van Dyke, "UriFrame: Web Uniform Resource Identifier Framework."
<http://www.neilvandyke.org/uriframe/>