Black Hole

Module system for Gambit

Bundled libraries documentation, revision 1. ** DRAFT **

Documentation distributed in PDF, OpenOffice and Microsoft Word versions.

Please email any feedback, questions and suggestions to the authors.

Table of Contents

Table of Contents, with all exports

Introduction

All of the bundled libraries are in the `std` module-resolver. I.e., to import SRFI 13 and 14, evaluate `(import (std srfi/13 srfi/14))`.

ds (data structures) directory

queue – FIFO queue

© Per Eckerdal

Not threadsafe.

(make-queue)

(queue-size queue)

(queue-push! queue element)

(queue-pop! queue)

(queue-empty? queue)

(queue-empty! queue)

(queue-front)

wt-tree – Weight balanced trees

See http://www.gnu.org/software/mit-scheme/documentation/mit-scheme-ref/Weight_002dBalanced-Trees.html .

(make-wt-tree-type)

(make-wt-tree)

(singleton-wt-tree)

(alist->wt-tree)

(wt-tree/empty?)

(wt-tree/size)

(wt-tree/add)

(wt-tree/delete)

(wt-tree/add!)

(wt-tree/delete!)

(wt-tree/member?)

(wt-tree/lookup)

(wt-tree/split<)

(wt-tree/split>)

(wt-tree/union)

(wt-tree/intersection)

(wt-tree/difference)

(wt-tree/subset?)

(wt-tree/set-equal?)

(wt-tree/fold)

(wt-tree/for-each)

(wt-tree/index)

(wt-tree/index-datum)

(wt-tree/index-pair)

(wt-tree/rank)

(wt-tree/min)

(wt-tree/min-datum)

(wt-tree/min-pair)

(wt-tree/delete-min)

(wt-tree/delete-min!)

(number-wt-type)

(string-wt-type)

misc directory

al – A-list helper library

exception – Exception handling helper library

mailbox – Single-element mailbox library

(make-empty-mailbox)

(mailbox-put!)

(mailbox-get!)

match – Erlang-style list matcher

(match value . handling-rules)

Example:

```
> (match '(1 2 3) ((1 2 3) #t) (else #f))
#t
> (match 3 (3 #t) (else else))
#t
> (match 4 (3 #t) (else else))
4
> (define (test msg)
  (match msg
    (('no-args)
      'no-args)
    (('one-arg a1)
      `("The arg is " ,a1))
    (('two-args a1 a2)
      `("The two args are " ,a1 ,a2))
    (else
      (list 'nothing-matched: else))))

> (test '(two-args 1 2))
("The two args are " 1 2)
> (test 'bob)
(nothing-matched: bob)
```

optionals – let-optionals special form support

© Olin Shivers

splice – Splicer

© Per Eckerdal

(splice . args)

Returns a conveniency splice data structure for containing the results of any splice process.

```
> (splice 1 2 3)
#<splice #2 data: (1 2 3)>
```

(unsplice-list list)

A very simple data structure that is intended for use with SXML and similar data structures. Use like:

```
(unsplice-list `(element ,(splice 1 2 3) 4)) => (element 1 2 3 4)
```

The point is that you should be able to write functions that return multiple elements.

```
> (define (a-fun) (splice 1 2 3))
> (define (b-fun) 4)
> (unsplice-list `(element ,(a-fun) ,(b-fun)))
(element 1 2 3 4)
```

u8v – U8vector utility functions

(u8vector . args)

```
> (u8vector 1 2 3)
#u8(1 2 3)
```

(u8vector-reverse v)

```
> (define u (make-u8vector 3)) (u8vector-set! u 1 1) (u8vector-set! U 2 2)
> > > (u8vector-reverse u)
#u8(2 1 0)
```

(subu8vector->hex-string u8vect start end), (u8vector->hex-string u8vect)

```
> (define u (make-u8vector 3)) (u8vector-set! u 1 1) (u8vector-set! u 2 2)
> > > (u8vector->hex-string u)
"000102"
```

(hex-string->u8vector s)

```
> (hex-string->u8vector "010203")
#u8(1 2 3)
```

(u8vector-invert! v)

```
> (define u (make-u8vector 3)) (u8vector-set! u 1 1) (u8vector-set! U 2 2)
> > > u
#u8(0 1 2)
> (u8vector-invert! u)
> u
#u8(255 254 253)
```

(hex-char->integer c)

Takes either of the characters 0-9 or a-f or A-F for argument, and returns the integer value 0-15.

```
> (map hex-char->integer '(#\0 #\a #\F))
(0 10 15)
> (hex-char->integer #\g)
*** ERROR IN "(generated)"@1.1 -- Character not 0-9, A-F, a-f. #\g
1>
```

uuid – UUID generator

(make-uuid)

Example:

```
> (make-uuid)
"DB7F49A3-BBF6-40F3-95A1-038CCACE1D88"
```

Uses Gambit's random number facility.

net directory

http-client – HTTP client

© Per Eckerdal

The library contains mechanisms to receive request responses streaming, see sourcecode.

(http-get-url uri #!optional http-username http-password (headers '()) (follow #t) query) , (http-post-url uri #!optional http-username http-password (headers '()) (follow #t) query)

Non-streaming HTTP GET/POST routines.

Return `(list response-code response-headers response-data)`.

http-common – Common HTTP features

© Marc Feeley, Per Eckerdal

http-server – HTTP server

© Marc Feeley

http-session – HTTP session helper library

© Per Eckerdal

Implementation of HTTP sessions, for use together with http-server.

(make-session-variable #!optional default-value)

In order to create a session variable, say `user-id`, do

```
(define user-id (make-session-variable [default value, defaults to #f]))
```

In the code that responds to HTTP requests, do `(user-id)` to get the value of the session variable, and `(user-id [new value])` to set its value.

THE CURRENT VERSION DOESN'T LET OVER OLD VALUES TO THE GARBAGE COLLECTOR, LEAKS 100% MEMORY.

tcpip – TCP/IP helper procedures

(call-with-tcp-client hostname port-num proc)

(ipv4->u8vector ip)

(tcp-socket-port->fd port)

(device-port-rdevice-condvar port)

(device-port-wdevice-condvar port)

(port-rtimeout port)

(port-wtimeout port)

(port-wait-for-input port)

(port-wait-for-output port)

uri – URI handling

(make-uri)

(uri?)

(uri-scheme)

(uri-scheme-set!)

(uri-authority)

(uri-authority-set!)

(uri-path)

(uri-path-set!)

(uri-query)

(uri-query-set!)

(uri-fragment)

(uri-fragment-set!)

(uri-userinfo)

(uri-host)

(uri-port)

(uri-query-string)

(parse-uri)

(parse-uri-query)

(string->uri)

(uri->string)

(string->uri-query)

(encode-for-uri)

(remove-dot-segments)

(uri-join)

(uri-join*)

x-www-form-urlencoded - x-www-form-urlencoded encoding and decoding

(encode-x-www-form-urlencoded fields)

(decode-x-www-form-urlencoded str)

(urlencode str)

(urldecode str)

(write-urlencoded str)

(write-x-www-form-urlencoded fields)

srfi directory

1 – List processing

See http://srfi.schemers.org/srfi-1/srfi-1.html .

13 – String

See http://srfi.schemers.org/srfi-13/srfi-13.html .

14 – Character-sets

See http://srfi.schemers.org/srfi-14/srfi-14.html .

16 – case-lambda special form support

Public domain.

See http://srfi.schemers.org/srfi-16/srfi-16.html .

19 – Time data types and procedures

See http://srfi.schemers.org/srfi-19/srfi-19.html .

95 – Sorting

By Richard A. O'Keefe. Public domain.

See http://srfi.schemers.org/srfi-95/srfi-95.html .

string directory

base64 – Base64 processing library

(base64-string->u8vector str) , (base64-substring->u8vector str start end)

(u8vector->base64-string u8vect #!optional (width 0)) , (subu8vector->base64-string u8vect start end #!optional (width 0))

digest – Hash computation library

(digest-file filename algorithm #!optional (result-type (digest-default-result-type)))

(digest-subu8vector)

(digest-u8vector)

(digest-substring)

(digest-string)

(digest-update-u32-be)

(digest-update-u32-le)

(digest-update-u16-be)

(digest-update-u16-le)

(digest-update-u8)

(get-zero-u8vector)

zero-u8vector

(digest-update-subu8vector)

(close-digest)

(open-digest)

(open-digest-sha-256)

(open-digest-sha-224)

(digest-update-sha-256)

(hash-block-init-sha-256)

(hash-block-init-sha-224)

(open-digest-sha-1)

(digest-update-sha-1)

(hash-block-init-sha-1)

(open-digest-md5)

(digest-update-md5)

(hash-block-init-md5)

(digest-update-block-digest)

(end-block-digest)

(process-last-block)

(convert-hash-block)

(block-digest-width-set!)

(block-digest-width)

(block-digest-big-endian?-set!)

(block-digest-big-endian?)

(block-digest-bit-pos-set!)

(block-digest-bit-pos)

(block-digest-block-pos-set!)

(block-digest-block-pos)

(block-digest-block-set!)

(block-digest-block)

(block-digest-hash-set!)

(block-digest-hash)

(block-digest-hash-update-set!)

(block-digest-hash-update)

(block-digest?)

(make-block-digest)

(hash-block->u8vector)

(hash-block->hex-string)

(open-digest-crc32)

(end-crc32)

(digest-update-crc32)

(crc32-table)

(crc32-digest-lo16-set!)

(crc32-digest-lo16)

(crc32-digest-hi16-set!)

(crc32-digest-hi16)

(crc32-digest?)

(make-crc32-digest)

(digest-state-set!)

(digest-state)

(digest-update-set!)

(digest-update)

(digest-end-set!)

(digest-end)

(digest?)

(make-digest)

pregexp – Posix Regular Expressions matching engine library

See http://www.ccs.neu.edu/home/dorai/pregexp/pregexp.html .

(pregexp s)

(pregexp-match-positions pat str . opt-args)

(pregexp-match pat str . opt-args)

(pregexp-split pat str)

(pregexp-replace pat str ins)

(pregexp-replace* pat str ins)

(pregexp-quote s)

(pregexp-match? pattern string)

sxml-to-xml – Fast SXML to XML generator

(sxml>>html* item #!optional (port (current-output-port)) maybe-level)

(sxml>>xhtml* item #!optional (port (current-output-port)) maybe-level)

(sxml>>xml* item #!optional (port (current-output-port)) maybe-level)

(sxml>>html-fast item #!optional (port (current-output-port)))

(sxml>>html item #!optional (port (current-output-port)))

(sxml>>xhtml-fast item #!optional (port (current-output-port)))

(sxml>>xhtml item #!optional (port (current-output-port)))

(sxml>>xml-fast item #!optional (port (current-output-port)))

(sxml>>xml item #!optional (port (current-output-port)))

(sxml>>pretty-xml-file item outpath #!optional noblanks)

(sxml>>pretty-xhtml-file item outpath #!optional noblanks)

(sxml->html-string-fragment item #!optional maybe-level)

(sxml->xml-string-fragment item #!optional maybe-level)

util – String utility procedures

(char->string char)

```
> (char->string #\a)
"a"
```

(string-strip str)

Removes all leading and ending `(char-whitespace?)`:s (that's a R5RS procedure).

```
> (string-strip "\tyo \r\n")
"yo"
```

(string-replace-char str from to)

```
> (string-replace-char "hullo" #\u #\e)
"hello"
```

(join between args)

```
> (join "a" '(1 2 3))
(1 "a" 2 "a" 3)
> (apply string-append (join " " '("a" "b" "c")))
"a b c"
```

(string-split chr str #!optional (sparse #f))

```
> (string-split #\space " wag the dog. ")
("" "wag" "the" "dog." "")
> (string-split #\space " wag the dog. " #t)
("wag" "the" "dog." "")
```

(string-split-at-first at-char s #!optional (not-found (lambda () (error "String parameter didn't contain char."))))

```
> (string-split-at-first #\space "wag the dog")
("wag" . "the dog")
> (string-split-at-first #\space "wag" (lambda () #f))
#f
> (string-split-at-first #\space "wag")
*** ERROR IN "(generated)"@1.1 -- String parameter didn't contain char.
1>
```

(string-split-at-first-nice at-char s)

```
> (string-split-at-first-nice #\space "wag the dog")
("wag" . "the dog")
> (string-split-at-first-nice #\space "wag")
#f
```

(string-name-split str)

```
> (string-name-split "Bob Doe")
("Bob" "Doe")
> (string-name-split "Bob     Doe ")
("Bob" "Doe")
> (string-name-split " Bob-Doe")
("Bob" "Doe")
```

(string-set-first-char! fun str)

```
> (define s "Test")
> (string-set-first-char! (lambda (c) (print "c is " c "\n") #\F) s)
c is T
"Fest"
> s
"Fest"
```

(string-capitalize! str) , (string-decapitalize! str)

Sets first character to its R5RS `char-upcase` and `char-downcase` correspondent, respectively.

```
> (define s "ttt")
> (string-capitalize! s)
"Ttt"
> s
"Ttt"
> (string-decapitalize! s)
"ttt"
> s
"ttt"
```

(string-constantize str) , (string-camelize str) , (string-dasherize str) , (string-spaceize str) , (string-humanize str)

```
> (define (test s)
    (for-each
      (lambda (f)
        (eval `(print "(" ',f "\t\"" ,s "\") => \"" (,f ,s) "\"\n")))
      '(string-constantize string-camelize string-dasherize
        string-spaceize    string-humanize)))
> (test "helloWorld")
(string-constantize     "helloWorld") => "HelloWorld"
(string-camelize        "helloWorld") => "helloWorld"
(string-dasherize       "helloWorld") => "hello-world"
(string-spaceize        "helloWorld") => "hello world"
(string-humanize        "helloWorld") => "Hello world"
> (test "hello-world")
```

```
(string-constantize     "hello-world") => "HelloWorld"
(string-camelize        "hello-world") => "helloWorld"
(string-dasherize       "hello-world") => "hello-world"
(string-spaceize        "hello-world") => "hello world"
(string-humanize        "hello-world") => "Hello world"
> (test "hello world")
(string-constantize     "hello world") => "HelloWorld"
(string-camelize        "hello world") => "helloWorld"
(string-dasherize       "hello world") => "hello-world"
(string-spaceize        "hello world") => "hello world"
(string-humanize        "hello world") => "Hello world"
```

The point of these procedures is to provide you with an easy way to benefit from multiple naming conventions.

For instance, in Scheme code, you may want variables `string-dasherize`:d (in-this-way), in a database, you may want column names to be in `camel-case` (inThisWay), and in error messages you may want them `string-humanize`:d (In this way).

(string-remove-suffix haystack needle) , (string-remove-prefix haystack needle)

```
> (string-remove-suffix "Sr Bob Sr" "Sr")
"Sr Bob "
> (string-remove-prefix "Sr Bob Sr" "Sr")
" Bob Sr"
```

(string-pluralize str #!optional (num 0)) , (string-depluralize str)

```
> (string-pluralize "Fish" 1)
"Fish"
> (string-pluralize "Fish" 2)
"Fishs"
> (string-depluralize "Fishes")
"Fishe"
> (string-depluralize "Boats")
"Boat"
```

(symbol-append . syms)

```
> (symbol-append 'a 'b 'c)
abc
```

(string-invert str) , (string-uninvert str)

The inverted form is hexadecimalized, utf8-encoded, inverted. This makes inverted strings sort in the reverse order of regular strings, both as they are or in utf8-encoded hexadecimalized form.

```
> (string-invert "\0abc")
"ff9e9d9c"
> (string-uninvert "ff9e9d9c")
"\0abc"
```

(dumps . args)

Produces a fair string representation of the arguments.

```
> (dumps 'a)
"a"
> (dumps "hi")
"hi"
> (dumps '(1 2 3))
"(1 2 3)"
> (dumps (make-u8vector 2))
"#u8(0 0)"
> (dumps "Two is " 2 " and a cons cell is " (cons 1 2))
"Two is 2 and a cons cell is (1 . 2)"
```

xml-to-sxml – Fast XML string to SXML parser

Uses the XML to SXML parser found in SSAX-SXML released public domain by Oleg Kiselyovs. Optimized for speed. Features the complete set of HTML entities.

html-entity-unicode-numbers

A-list of HTML entity references and their corresponding unicode numbers, i.e. '(amp . 38) etc.

html-entity-unicode-chars

Same as html-entity-unicode-numbers, but car is string rather than symbol, and cdr is a one character long string containing the unicode character, rather than the its integer unicodenumber.

(xml-string->sxml xml-string #!optional (namespace-prefix-assig '()))

Index