generator egg

Provides generator-like coroutine operations.
Extension for Chicken Scheme
Version 1.1

Kon Lovett

Table of Contents

1 About this egg

1.1 Version history

`1.1` Added generator procedure

`1.0` Initial release

1.2 Usage

Load this egg like so:

```
(require-extension generator)
```

2 Documentation

generator is a set of routines supporting generator-like operations.

`coroutine` [macro]

```
(coroutine ARG EXPR ...)
```

Returns a coroutine procedure of one argument. Use '(resume COROUTINE VALUE)' within the body to invoke another coroutine.

`define-generator` [macro]

```
(define-generator (NAME ARG ...) EXPR ...)
```

Defines a procedure, `NAME`, that, when invoked, returns a generator parameterized on '`ARG ...`'. The '(yield VALUE)' procedure is used within the body '`EXPR ...`' of the generator to produce a single value. Multi-valued yield is not supported.

`make-generator` [procedure]

```
(make-generator ROUTINE [SENTINEL])
```

Returns a generator procedure, of zero or one argument.

`ROUTINE` is a procedure of one argument, invoked with the generator "yield value" procedure. The yield procedure takes one argument, the "next value".

`SENTINEL` is a value to return, or a zero-argument procedure to call, when the generator becomes 'dead', otherwise continues at caller parent.

The generator procedure argument may be:

none - invokes the routine `ROUTINE` with "yield value" procedure

'status - returns the generator run status, 'dead or 'alive

'dead? - queries whether generator is not runnable

'alive? - queries whether generator is runnable

'kill! - makes the generator not runnable

`generator` [procedure]

```
(generator PROC-OF-YIELDER [SENTINEL])
```

Takes a one argument procedure to generate. Invokes the generating procedure with the 'yield' procedure as the argument. The generating procedure calls the yield procedure to produce a value. The generator terminates when the generating procedure exits without invoking yield.

When the `SENTINEL` is missing 'end-of-generator is used.

3 Examples

```
(use generator)

(define (list->iterator ls) (make-generator (lambda (yield) (for-each yield ls))))

(define (iterator-empty? lsi) (lsi 'dead?))

(define t (list->iterator '(1)))
(t)             # 1; continues at caller
(t)             # nothing; continues at caller parent
(t)             # Error: (generator) dead

(define t (list->iterator '(1 2)))
(t)             # 1; continues at caller
(t 'kill!)      # dead
(t)             # Error: (generator) dead

(define-generator (counter n) (yield n) (counter (add1 n)))
(define t (counter 20))
(t)     # 20
(t)     # 21
...and so on

(define (gen-list lst)
  (generator
    (lambda (yield)
      (let loop ([lst lst])
        (when (pair? lst)
           (yield (car lst))
           (loop (cdr lst)))))))

> (define gen (gen-list '(a b c)))
> (gen)
a
> (gen)
b
> (gen)
c
> (gen)
end-of-generator
```

4 License

Copyright (c) 2005, Kon Lovett. All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the Software), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED ASIS, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Index