Basic Scheme Macros for Function Docs

Richard Cleis
ABQ Lisp/Scheme Users Group
Albuquerque, New Mexico
December 16, 2007
A Few dozen pages to describe the following...

(lam-w/args args body)
(lam-w/expr args body)
(lam-w/info info args body)

(docs-for function)
(list-docs function1 [function2 ...])

... yet they are implemented in one page

using only elementary techniques of Scheme and define-syntax
Utilized Macro Features

S-Expressions are handled, unevaluated, with names

(arg . body) handles the lambda form, which is 'implied-begin'

Ellipses represent the names of multiple input functions

Nameless procedures have temporary names in macroland

Syntax-error is used as described in define-syntax-primer.txt
Means for associating information with lambda expressions

High level programming often requires quick reference

Slight variations of lambda fill a table of expressions

Examples: 3 similar macros

Lambda implementation is leached for args and body

Conclusions: Scheme + syntax-language... effective
Means for associating information with lambda expressions

Intended for high level functions in workspaces

- High level programming often requires quick reference
- Slight variations of lambda fill a table of expressions
- Examples: 3 similar macros
- Lambda implementation is leached for args and body
- Conclusions: Scheme + syntax-language... effective
Means for associating information with lambda expressions

High level programming often requires quick reference

- Digging through source isn't always practical

Slight variations of lambda fill a table of expressions

Examples: 3 similar macros

Lambda implementation is leached for args and body

Conclusions: Scheme + syntax-language... effective
Means for associating information with lambda expressions

High level programming often requires quick reference

Slight variations of lambda fill a table of expressions

A few macros and functions access the expressions

Examples: 3 similar macros

Lambda implementation is leached for args and body

Conclusions: Scheme + syntax-language... effective
Means for associating information with lambda expressions

High level programming often requires quick reference

Slight variations of lambda fill a table of expressions

Examples: 3 similar macros

May reference functions or expressions

Lambda implementation is leached for args and body

Conclusions: Scheme + syntax-language... effective
Means for associating information with lambda expressions

High level programming often requires quick reference

Slight variations of lambda fill a table of expressions

Examples: 3 similar macros

Lambda implementation is leached for args and body

Function reference (not name) provides key

Conclusions: Scheme + syntax-language... effective
Means for associating information with lambda expressions

High level programming often requires quick reference

Slight variations of lambda fill a table of expressions

Examples: 3 similar macros

Lambda implementation is leached for args and body

Conclusions: Scheme + syntax-language... effective

Many solutions can easily be investigated
Means for associating information with lambda expressions

High level programming often requires quick reference

Slight variations of lambda fill a table of expressions

A few macros and functions access the expressions

Examples: 3 similar macros

Lambda implementation is leached for args and body

Conclusions: Scheme + syntax-language... effective
How to doc lambda expressions

Three macros fill doc table; two are just like lambda:

(lam-w/args args body)
(lam-w/expr args body)
(lam-w/info info args body)

A function, docs-for, extracts the docs

(docs-for function)

A macro can be used to associate names to docs

(list-docs function1 [function2 ...])

Docs-for works with any name for a function (including none)
List-docs lists the docs with any name for the function
Means for associating information with lambda expressions

High level programming often requires quick reference

Slight variations of lambda fill a table of expressions

Examples: 3 similar macros

May reference functions or expressions

Lambda implementation is leached for args and body

Conclusions: Scheme + syntax-language... effective
Macro that saves expression for the args

Often, carefully named args are enough to 'jog your memory'

```
(define interest-a
  (lam-w/args (principal percent)
    (* principal 0.01 percent)))
```

To get the docs, use:

```
> (docs-for interest-a) ; produces...
(lambda (principal percent))
```

To associate a name:

```
> (list-docs interest-a) ; produces...
(interest-a (lambda (principal percent)))
```
Macro that saves the expression for the entire function

High level functions may be short; lam-w/expr can doc them

```scheme
(define interest-e
  (lam-w/expr (principal percent)
    (* principal 0.01 percent)))

> (docs-for interest-e) ; produces...
(lambda (principal percent)
  (* principal 0.01 percent))
```
Macro that saves the expression for anything

Sometimes, a string is desired (Any legal expression is ok)
• Docs-for will evaluate the info expression if it is a procedure
• Nature is upset, though; lambda now has info, args, and a body

```
(define interest-i
  (lam-w/info "Supply principal & rate as percent"
    (principal percent)
    (* principal 0.01 percent)))

> (docs-for interest-i)
"Supply principal & rate as percent"
```
Macro that saves the expression for a function

A function in the first arg can do anything
• It evaluates whenever docs-for is used
• The arg may be a function name (below, it's embedded)
• GUI's and talking docs are easy to implement

(define interest-f
  (lam-w/info (lambda ()
    (display "This could feed a gui")
    (newline)
    (principal percent)
    (* principal percent)))

> (docs-for interest-f)
"This could feed a gui"
List-docs macro associates names with docs

A function would need the names and the procedures
• Scheme extracts a name with 'tick... frightens Schemeaphobics
• A list of pairings is normally needed, anyway... ergo: this macro

> (list-docs interest-a
   interest-e
   interest-i
   interest-f)
((interest-a (lambda (principal percent)))
 (interest-e
   (lambda (principal percent)
     (* principal 0.01 percent)))
 (interest-i "Supply principal & rate as percent")
 (interest-f #<procedure>))
All previous examples listed as nameless procedures:

(define list-of-functions
  (list
    (lam-w/args (principal percent)
      (* principal 0.01 percent))
    (lam-w/expr (principal percent)
      (* principal 0.01 percent))
    (lam-w/info "Supply principal & rate as percent"
      (principal percent)
      (* principal 0.01 percent))
    (lam-w/info (lambda ()
      (display "This could feed a gui")
      (newline))
      (principal percent)
      (* principal 0.01 percent)))))
Docs-for works with the list of nameless procedures

The four procedures only appear identical in this REPL
All are actually distinct keys to the docs...

> list-of-functions
(#<procedure:f>
  #<procedure:f>
  #<procedure:f>
  #<procedure:f>)
Docs-for naturally maps the nameless list

> (map docs-for list-of-functions)
This could feed a gui
((lambda (principal percent))
 (lambda (principal percent)
   (* principal 0.01 percent))
"Supply principal & rate as percent"
#<void>)

The doc of the last procedure returns void after displaying the text

Function synonyms work, too
• Function names contain a procedure just like an element of a list
Means for associating information with lambda expressions

High level programming often requires quick reference

Slight variations of lambda fill a table of expressions

Examples: 3 similar macros

Lambda implementation is leached for args and body

Function reference (not name) provides key

Conclusions: Scheme + syntax-language... effective
Macros copy expressions in lambda and table the docs

The temporary f is formed in syntax space
• f is used for the key and the Macro's final value.

```
(define-syntax lam-w/args
  (syntax-rules ()
    ((_ args . body)
      (let ((f (lambda args . body)))
        (sort-table! f '(lambda args))
        f))
    ((_ ...) (syntax-error "Use λ form. Expr for args will be tabled."))))
```

lam-w/args replaces the sort line with:

```
(sort-table! f '(lambda args . body))
```
A simple list and one function for making docs

Any 'real' sorter or hash-table could be used
• yet a textbook list works, and is silly-simple.

```
(define  table (list))
(define  sort-table!
  (lambda  f&info
    (set!  table (cons f&info table))))
(define  (get-info f table)
  (cond  ((null?  table)
    #f)
    ((equal? f (caar table))
      (cadar table))
    (else
      (get-info f (cdr table))))
```
docs-for evaluates a procedure if it gets one

docs-for is defined with lam-w/args
• (docs-for docs-for) returns the expression for its args
• Programming has no soul if such things are not kewl

```
(define docs-for
  (lam-w/args (f-as-key)
    (do-info (get-info f-as-key table))))
(define do-info
  (lambda (info)
    (cond (info (cond ((procedure? info) (info))
                     (else info)))
          (else 'not-in-table)))
```
The list-docs macro lists a name with the docs

To be used for making help tables in GUIs, for example
• The macro allows 'ticking' the procedure name, and
• forming the list with ellipses

```
(define-syntax  list-docs
  (syntax-rules ()
    ((_ f ...) (list (list 'f (get-info f table)) ...))
    ((_ ...) (syntax-error "Use (list-docs f [...])")))
```
lam-w/info expects an expression, info, before args

Docs-for will evaluate info if a function, return info otherwise

```
(define-syntax lam-w/info
  (syntax-rules ()
    ((_ info args . body)
      (let ((f (lambda args . body)))
        (sort-table! f info)
        f))
    ((_ ...) (syntax-error "Use λ form, but put info before args.")))
```
Macro syntax-error from define-syntax-primer.txt

Causes REPL to provide information about the macro
- Designed to work with recursive macros
- Designed to fail so that caller's text is returned
- Programming has no soul if... oh, never mind

```
(define-syntax syntax-error
  (syntax-rules ()
    ((syntax-error)
      (syntax-error "Bad use of syntax error!")
    )))
```
Means for associating information with lambda expressions

High level programming often requires quick reference

Slight variations of lambda fill a table of expressions

Examples: 3 similar macros

Lambda implementation is leached for args and body

Conclusions: Scheme + syntax-language... effective

Many solutions can easily be investigated
Conclusions other than: this solution is practical

Macros permit matching multiple patterns and recursion, but in this case it is easier to read, debug, and use a macro for each form.

Basing these macros on lambda (instead of define) is more flexible
• E.g. when using synonyms and lists of unnamed functions

Possible non-portable improvements
• Use hash-tables or sorting functions to table information
• Contain code in a Scheme Module (like this implementation)
• Use GUI to display and manipulate the information

The reasons for define-syntax are learned by doing, not reading.