

args egg

Command-line argument handling facilities, layered on SRFI 37 (args-fold).
Extension for Chicken Scheme
Version 1.0

Zbigniew

Table of Contents

1	About this egg	1
1.1	Version history	1
1.2	Requirements	1
1.3	Usage	1
2	Documentation	2
2.1	Creating options	2
2.2	Parsing the command line	2
2.3	Usage information	3
2.4	Operands and unrecognized options (advanced)	4
3	Bugs	5
4	Examples	6
5	License	8
	Index	9

1 About this egg

1.1 Version history

1.0 Initial release

1.2 Requirements

This egg requires the following extensions:

```
srfi-37 [args-fold], srfi-13 [string-lib], srfi-1 [list-lib]
```

1.3 Usage

Load this egg like so:

```
(require-extension args)
```

2 Documentation

This extension provides a wrapper around SRFI 37 (`args-fold`). The main goal is to let the user parse command-line arguments without having to write a lot of similar support code every time.

By default, options and operands (non-options) are collected into two lists and returned by the parser, and unrecognized options complain and display help. Therefore, it is very possible not to write any option-procs, operand-procs, or unrecognized-procs as required by SRFI 37. However, the capability to customize is there should you need it.

Additionally, the help text for your options can be generated for you, so your options and usage information don't get out of sync.

2.1 Creating options

`args:make-option` [macro]

`(args:make-option (OPTION-NAME ...) ARG-DATA [BODY])`

Make an `args:option` record, suitable for passing to `args:parse`.

`OPTION-NAME ...` is a sequence of short or long option names. They must be literal symbols; single-character symbols become short options, and longer symbols become long options. So `(args:make-option (c cookie) ...)` specifies a short option `-c` and long option `-cookie`. Under the hood, `(c cookie)` becomes `'(#\c "cookie")`, as expected by SRFI 37's `OPTION`.

`ARG-DATA` is either a pair `(ARG-TYPE ARG-NAME)` or a plain keyword `ARG-TYPE`. `ARG-TYPE` is a keyword that specifies whether the option takes an argument:

`#:required`

Argument is required

`#:optional`

Argument is optional

`#:none`

No argument (actually, any other value than `#:required` or `#:optional` is interpreted as `#:none`)

`ARG-NAME`, if provided, is a string specifying the name of the argument. This name is used in the help text produced by `args:usage`.

`BODY` is an optional sequence of statements executed when this option is encountered. Behind the scenes, `BODY` is wrapped in code which adds the current option and its argument to the final options alist. So, simply leave `BODY` blank and options will be collected for you. `BODY` is an option-processor as defined in SRFI 37, and has access to the variables `OPT` (the current `#<option>`), `NAME` (the option name) and `ARG` (argument value or `#f`).

2.2 Parsing the command line

`args:parse` [procedure]

`(args:parse ARGS OPTIONS-LIST [OPTIONALS])`

Parse `ARGS`, a list of command-line arguments given as strings, and return two values: an alist of option names (symbols) and their values, and a list of operands (non-option arguments).

Operands are returned in order, but options are returned in reverse order. Duplicate options are retained in the options alist, so this lets `ASSQ` find the *last* occurrence of any duplicate option on the command line. A (name . value) pair is added for each alias of every option found, so any alias is a valid lookup key.

`OPTIONS-LIST` is a list of accepted options, each created by `args:make-option`.

`OPTIONALS` is an optional sequence of keywords and values:

```
#:operand-proc PROC
      calls PROC for each operand, with arguments OPERAND OPTIONS
      OPERANDS
```

```
#:unrecognized-proc PROC
      calls PROC for each unrecognized option, with arguments OPTION
      NAME ARG OPTIONS OPERANDS
```

The default `operand-proc` is a no-op, and the default `unrecognized-proc` issues an error message and calls the help option's processor. See the `args-fold` documentation for usage information and an explanation of the procedure arguments; `OPTIONS` and `OPERANDS` are seed values.

`args:help-options` [parameter]

List of option names (strings or single characters, as in `SRFI 37`) to be considered 'help' options, in order of preference. `args:parse` uses this to select a help option from the option list it is passed. This is currently used only for unrecognized options, for which the help option is automatically invoked.

By default, `-help`, `-h` and `-?` are considered help options.

2.3 Usage information

Well-behaved programs display help or usage text when invoked with an option such as `-help`. `args:usage` will generate a formatted list of options in the GNU style, from a list of `args:options`. Around this you might place a descriptive header and footer.

`args:usage` [procedure]
`(args:usage OPTION-LIST)`

Generate a formatted list of options from `OPTION-LIST`, and return a string suitable for embedding into help text. The single string consists of multiple lines, with a newline at the end of each line. Thus, a typical use would be `(print (args:usage opts))`.

`args:width` [parameter]

We don't auto-format the left column (the option keys) based on the length of the longest option, but you can override it manually. Example:

```
(parameterize ((args:width 40)) (args:usage opts))
```

2.4 Operands and unrecognized options (advanced)

These are suitable for use with `#:operand-proc` or `#:unrecognized-proc` in `args:parse`. Most users will probably not customize these procedures themselves, but a couple useful prefabricated ones are provided.

`args:ignore-unrecognized-options` [procedure]
Silently ignore unrecognized options, and omit from the options alist.

`args:accept-unrecognized-options` [procedure]
Silently add unrecognized options to the options alist.

`args:make-operand-proc` [macro]
(`args:make-operand-proc` [BODY])

Return a procedure suitable for using as an operand procedure in `args:parse`. Provides the arguments `OPERAND`, `OPTIONS`, and `OPERANDS` to the `BODY`; where `OPERAND` is the current operand (as in `args-fold`) and `OPTIONS` and `OPERANDS` are `SEEDS` (as in `args-fold`) and should not be modified. Also wraps `BODY` in code that adds the operand to the final operand list (seed).

3 Bugs

The name `args:make-option` is verbose.

4 Examples

```
(use args)

(define opts
  (list (args:make-option (c cookie)    #:none      "give me cookie"
    (print "cookie was tasty"))
    (args:make-option (d)              (optional: "LEVEL") "debug level [default: 1]")
    (args:make-option (e elephant)     #:required  "flatten the argument"
    (print "elephant: arg is " arg))
    (args:make-option (f file)        (required: "NAME")  "parse file NAME")
    (args:make-option (v V version)   #:none      "Display version"
    (print "args-example $Revision: 1.3 $")
    (exit))
    (args:make-option (abc)           #:none      "Recite the alphabet")
    (args:make-option (h help)        #:none      "Display this text"
    (usage))))))

(define (usage)
  (with-output-to-port (current-error-port)
    (lambda ()
      (print "Usage: " (car (argv)) " [options...] [files...]")
      (newline)
      (print (args:usage opts))
      (print "Report bugs to zbigniewsz at gmail.")))
  (exit 1))

(receive (options operands)
  (args:parse (command-line-arguments) opts)
  (print "-e -> " (alist-ref 'elephant options))) ;; 'e or 'elephant both work

;; If command line is --cookie -e test -e hello:
;; cookie was tasty
;; elephant: arg is test
;; elephant: arg is hello
;; -e -> hello

;; If command line is --cookie -e test --foo:
#|
cookie was tasty
elephant: arg is test
./args-example: unrecognized option: foo
Usage: ./args-example [options...] [files...]

-c, --cookie          give me cookie
-d [LEVEL]            debug level [default: 1]
-e, --elephant=ARG    flatten the argument

```

```
-f, --file=NAME      parse file NAME
-v, -V, --version    Display version
  --abc              Recite the alphabet
-h, --help           Display this text
```

Report bugs to zbigniewsz at gmail.

|#

Additional examples can be found in args-examples.scm.

5 License

Copyright (c) 2005, 2006 Jim "Zb" Ursetto. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the name of the author nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Index

args:accept-unrecognized-options.....	4	args:make-option.....	2
args:help-options.....	3	args:parse.....	2
args:ignore-unrecognized-options.....	4	args:usage.....	3
args:make-operand-proc.....	4	args:width.....	3