File No. UIUCDCS-F-85-937

MACROS AND UTILITIES

FOR

FRANZ LISP

Jeff Becker

Department of Computer Science
University of Illinois
at Urbana-Champaign

February 1985

ISG 85-7

This work was supported in part by the National Science Foundation under grant NSF DCR 84-06801, and the Office of Naval Research under grant ONR N00014-82-K-0186.

Table of Contents

MACROS AND UTILITIES FOR FRANZ LISP

1. Introduction

This manual describes a package of macros and utilities for Franz Lisp [Foderaro, Sklower, and Layer, 1983] which make writing programs easier, and make it easier to translate from Interlisp [Teitelman et al, 1978] to Franz Lisp. The facilities for structure declaration, formatted output, and iteration also help to make Lisp code more readable and maintainable. This is a working document intended to make these facilities more widely available. It is intended for use by programmers familiar with Franz Lisp and uses the notation established in the Franz Lisp Manual for giving a lot of information about function arguments in a short space.

All files except util.l and comp.l exist in both interpreted and compiled form. The package consists of the following files :

FILE	CONTENTS
macros.l	miscellaneous small but useful macros
iterate.l	iterative control structure macros
struct.l	simple structure declaration package
printout.l	output formatting macro
listfns.l	miscellaneous list manipulations (lambdas)
hash.l	hash table functions
getf.l	frequency ordered disembodied property list "get"
sysfn.l	useful system calls package
clear.c	device independent screen clear
util.l	loads the entire utilities package
comp.l	loads files needed for compiling

Object files are located in these directories:

on Vax A&B <path> = /usr/lib/lisp/local/jbecker

on aisun* <path> = /ailib/lisplib

2. Accessing the Files

One may load individual files from the package into the Lisp interpreter as desired. To load the entire package, put the following line in a .lisprc file in your home directory. This will load util.l, which will in turn load each of the files in the utilities package (substitute the path given above for the appropriate machine):

```
(load '<path>/util.l)
```

The file util.l contains the following :

```
(load '<path>/macros)
(load '<path>/listfns)
(load '<path>/iterate)
(load '<path>/struct)
(load '<path>/hash)
(load '<path>/getf)
(cfasl '<path>/clear.o '_clear 'clear "subroutine")
(load '<path>/printout)
(load '<path>/sysfn)
```

3. Using the Franz Compiler

When using the Liszt compiler with this package, one may selectively load the needed files, or load comp.l which contains the following :

```
(load '<path>/macros)
(load '<path>/listfns)
(load '<path>/iterate)
(load '<path>/struct)
(load '<path>/printout)
```

Some special declarations are needed when compiling certain functions. When "time" is used, include the following expression in your source file:

```
(declare (*fexpr time))
```

When "structure" is used, enclose the structure declaration as follows :

```
(eval-when (eval compile) (structure 'TypeName '(f1 f2 ...)))
```

4. macros.l

Most of the functions in this file are implemented as macros. These functions support generalized assignment, stack operations, queue operations, boolean operations, and others.

4.1. (:= 'g_refexpr 'g_value)

Generalized structure assignment operator based on the "setf" macro (Section 2 of the Franz Lisp manual). Allows abbreviated reference to left hand side by "*-*" in the right hand side. Returns g_value.

4.2. (newstack s_arg)

Sets s_arg to be an empty stack. A variable should be initialized to be an empty stack using newstack before performing any other stack operations with it.

4.3. (push 'g_arg 'l_arg)

Push element g_arg onto the head of the stack l_arg. This is an undocumented built-in macro.

4.4. (pop 'l_arg)

Returns the top element of l_arg and sets l_arg to be the remaining elements. This is an undocumented built-in macro.

4.5. (top 'l_arg)

Returns the top element of stack l_arg without changing the stack.

4.6. (newqueue s_arg)

Sets s_arg to be a new, empty queue. A variable should be initialized to be an empty queue using newqueue before performing any other queue operations with it.

4.7. (enqueue 'g_arg 'l_ptr)

Add element g_arg to the end of a queue, where l_prt is a queue.

4.8. (dequeue 'l_ptr)

Removes and returns the first element from the queue, where l_ptr is a queue.

4.9. (queue-empty? 'l_ptr)

Returns non-nil if the given queue is empty.

4.10. (queuetolist 'l_ptr)

Returns a list given a queue.

4.11. (listtoqueue 'l_arg)

Returns a queue given a list.

4.12. (put 'ls_name 'g_ind 'g_val)

A variation of putprop, with the second and third arguments reversed.

4.13. (vput 'v_name 'g_ind 'g_val)

A variation of vputprop with the second and third arguments reversed.

4.14. (nconc1 'l_arg 'g_arg)

Add element g_arg to the end of list l_arg.

4.15. (consend 'l_arg 'g_arg)

Add element g_arg to the end of list l_arg. (Same as nconc1).

4.16. (consp 'g_arg)

The same as (dtpr 'g_arg). Returns t if g_arg is a non-nil list cell.

4.17. (logor 'x_arg1 'x_arg2)

Returns the logical *or* of its arguments.

4.18. (logand 'x_arg1 'x_arg2)

Returns the logical *and* of its arguments.

4.19. (logxor 'x_arg1 'x_arg2)

Returns the logical *exclusive or* of its arguments.

4.20. (logdiff 'x_arg1 'x_arg2)

Returns the logical *difference* of its arguments (bitclear).

4.21. (rsh 'x_val 'x_amt)

Returns x_val right shifted x_amt places. If x_amt is negative, then x_val is left shifted.

4.22. (onbits 'x_val)

Returns the number of set bits in fixnum x_val. When fixnums are used to represent sets, this function is useful in determining set cardinality. Takes time proportional to the number of set bits.

5. Iterate.l

This file contains the iterative control structure macros described below. The *while* and *repeat* macros may be used for simple side-effect operations. The *for* macro may be used for generating a variety of return values, and for simultaneous iterative sequences.

5.1. (while 'g_pred do 'g_exp1 ...)

A basic while loop macro. While g_pred evaluates to non-nil, the expressions following "do" are evaluated (a typical while loop). This expands into a prog. Always returns nil.

5.2. (repeat 'g_exp1 ... until 'g_pred)

A basic repeat loop macro. The expressions between "repeat" and "until" are evaluated, and if g_pred evaluates to non-nil, it is repeated (a typical repeat loop). This expands into a prog. Always returns nil.

5.3. (for <init/repeat/terminate>... i.s.opr <body>)

This is a flexible iterative statement construction macro. "i.s.opr" stands for "iterative statement operator". It specifies the operation to be done on each iteration. An "ivar" is a local variable which is reassigned on each iteration. The full syntax is:

```
(for <init/repeat/terminate> [as <init/repeat/terminate> ...]
    i.s.opr
    [when  <cond>]
    [while <cond>]
    [body]
    [until <cond>] )
```

where **<init/repeat/terminate>** is:

```
ivar in   list [by <fn>]                          ; map by elements
ivar on   list [by <fn>]                          ; map by tails
ivar from <number> [to <number>] [by <number>]    ; counting loop
```

body
: is a list of expressions, optionally accompanied by "when", "while" and "until" forms. The loop body is evaluated as if it were contained in a "progn".

when
: causes evaluation of the body to be skipped for one iteration if the condition is not satisfied. The test is always done before the body is evaluated.

while
: causes a loop exit if the condition is not satisfied. The test is always done before the body is evaluated.

until
: causes a loop exit if the condition is satisfied. The test is always done after the body is evaluated.

exitif
: causes a loop exit if the condition is satisfied. The test is done in the position in the body at which it occurs.

Provided iterative statement operators:

do	for side effects only, returns nil
collect	make a list of the results obtained by evaluating body
splice	nconc the results obtained by evaluating body
append	non-destructive form of splice
filter	make a list of the non-nil results
sum	sum results of evaluating body
count	count number of times body evaluates to non-nil
product	multiply results of evaluating body
average	the average of the results of evaluating body
always	return t if body always evaluates to non-nil
never	return t if body never evaluates to non-nil
exists	return t if body ever evaluates to non-nil
find	return result of first non-nil evaluation of body

A "for" loop expands into one of the Lisp "mapping" function when a *do, collect,* or *splice* iterative statement operator is used with an *in* or *on* list traversal form, and no *when, while, until,* or *exitif* conditions are used. Otherwise, it expands into a prog as follows:

```
(prog (ivars)
                {init forms}      initialize iterative variables
  //loop
                {test1 forms}     exit if any predicate evaluates to true
                {whentest}        skip body if this evaluates to false
                {body}            apply i.s.opr to body
  //iterate
                {test2 forms}     exit if any predicate evaluates to true
                {update forms}    update iterative variables
  //out
                {final forms})    finalize (usually a return)
```

Here are some simple examples of what can be done with the "for" macro ...

Finding a particular subset of a list:

(for x in '(a b c) collect x when (neq x 'a)) returns (b c)

Splicing sublists of a list:

(for x in '((a b)(c d e f)(g)) splice x) returns (a b c d e f g)

Combining two lists:

(for x in '(a b c) as y in '(d e f) collect (list x y)) returns ((a d)(b e)(c f))

Collect every other element of a list:

(for x in '(a b c d e) by cddr collect x) returns (a c e)

6. struct.l

This package provides a method of easily defining structures with named fields. Declaring a structure defines accessing macros and functions for creating new instances of a structure type.

6.1. (structure 's_name '(s_field1 ....))

Defines simple structures, where a structure is a list of named fields. The declaration format is

(structure 'typename '(field1 field2 ...)).

Structure defines a make- function, which returns a list of nil's of the appropriate size, a build- function which makes an instantiated structure given fillers for each field, and accessing macros for each field. Accessing macros are defined using the given field names, so be careful of conflicting names. A list is used to represent small structures (< 8 fields) and a vector is used to represent larger structures. Assignment may be accomplished with the "set-<field>" macros created when the structure is declared, or one may use the ":=" operator.

For example:

(structure 'box '(height width color))

defines macros:

make-box, build-box, height, width, color, set-height, set-width, and set-color.

Executing:

(setq box1 (build-box 5 4 'red))

sets the value of box1 to the list (5 4 red).

Then:

(height box1) returns 5,

(color box1) returns red.

(set-height box1 9) returns 9, and modifies the height field of box1.

Note: to compile, use

(eval-when (eval compile) (structure 'TypeName '(F1 F2 ...)))

7. printout.l

This is a printout package for Franz Lisp. It is modelled loosely after the Interlisp CLISP printout command. It allows printing of multiple arguments, with some formatting commands.

7.1. (printout g_arg1 ...)

Each g_arg may be either a special name recognized by "printout", or something to be printed. If a special name expects a(n) argument(s), the next item(s) in the list of arguments is(are) used. This expands into a progn if more than one Lisp expression is needed.

Implemented Printout Forms:

.clear	clear the screen and home the cursor
.drain	drain the current output port
.f format value	use cprintf to print value using given format
.l list separator	print list without parens using separator
.p list	apply printout to a list of printout forms
.pp form	do (pp-form form)
.sp n	output n spaces
.skip n	skip n lines
.tab n	tab to nth column (wraps if necessary)
.to port	direct subsequent output to the given port
t	print one newline (terpri)
* (form)	evaluate (form) exactly as given
-default-	"princ" anything else

Other useful printing utilities:

7.2. (space x_arg [p_port])

Print n spaces to the given port, or the default output port.

7.3. (skip x_arg [p_port])

Print n newlines to the given port, or the default output port.

7.4. (prinlist 'l_arg 's_separator [p_port])

"princ" each item in a list, with the given separator between each item, to the given port, or the default output port.

7.5. (clear)

Device independent screen clear. This is loaded in util.l.

8. llstfns.l

These are some extra list functions that Franz Lisp lacks. These are all defined as expr's (lambdas) rather than macros. These functions are non-destructive unless indicated otherwise.

8.1. (union 'l_arg1 'l_arg2)

Returns the list l_arg2 with all of the elements from l_arg1 which are not members of l_arg2 cons'ed to the front. Union is more efficient if l_arg1 is shorter. Union is non-commutative, that is, if l_arg2 contains an element which appears more than once, it will appear more than once in the value of union.

8.2. (lntersection 'l_arg1 'l_arg2)

Returns a list who's elements are members of both l_arg1 and l_arg2.

8.3. (ldifference 'l_arg1 'l_arg2)

Returns a list of all elements of l_arg1 that are not members of l_arg2.

8.4. (makeset 'l_arg)

Returns a list with all duplicate members removed.

8.5. (subset 'l_arg1 'l_arg2)

Returns t if the set represented by list arg1 is a subset of the set represented by list arg2.

8.6. (ldlsjoint 'l_arg1 'l_arg2)

Returns t if the sets represented by lists arg1 and arg2 are disjoint.

8.7. (firstn 'l_arg 'x_count)

Returns a list of the first n elements of l_arg.

8.8. (lastn 'l_arg 'x_count)

Returns (cons (firstn l_arg k)(nthcdr k l_arg)), where k = (length l_arg) - x_count. For example, (lastn '(a b c d e) 2) returns ((a b c) d e).
Note: this does not work exactly like the Interlisp version when x_count > (length l_arg).

8.9. (rotate 'l_arg)

Returns 'l_arg rotated one place so that the second element becomes the head, and the first element becomes the last element. Actually modifies 'l_arg.

8.10. (flatten 'l_arg)

Return a list of atoms which are obtained by a depth first traversal of the (possibly) nested list l_arg. For example, (flatten '(a (b c (d e)) f)) returns (a b c d e f).

8.11. (makename 's_leader)

Creates a name using ALL of 's_leader as a leader followed by an integer number with no leading zeros. The new name is not interned. If leader is nil, all name counters are reset. If leader is a list, the counter associated with the head of list is reset.

9. hash.l

Hash tables are most useful for large sets of properties (> 10). Hash tables are just like fast property lists. The syntax for commands puthash, gethash, and remhash are the same as for get, put, and remprop, except that a hash table instead of a symbol is the handle. Note that a hash table must be created using new-hash-table before it can be used. Keys must be "eq" not just "equal".

A bucket hash is used. Buckets are frequency ordered, so that a window on the "working set" of names is maintained in the first position in each bucket. (see getf.l).

9.1. (new-hash-table 'x_size)

Creates and returns a new hash table of >= given size. A semi-prime number is used as the table size. The table is a vector, and each table entry is a disembodied property list.

9.2. (puthash 'table 'g_key 'g_value)

Enters a value in the given hash table under the given key.

9.3. (gethash 'table 'g_key)

Gets the value from the given hash table under the given key.

9.4. (remhash 'table 'g_key)

Removes the given key and associated value from the given table.

9.5. (hashstatus 'table)

Return a list of fixnums representing the number of entries in each hash table bucket. Ideally there should be the same number of entries in each bucket.

10. getf.l

This file contains the single function which is used for maintaining frequency ordered disembodied property lists. Whenever a property is accessed, it is moved up to the front of the list. In this way, the most frequently accessed properties will tend to reside near the front of the list, making access quicker. A hand optimized assembly version of this function resides in getf.s.

10.1. (getf 'l_plist 's_property)

Just like regular property list get, except that the property list will be rearranged (for disembodied property lists only).

11. sysfn.l

This file contains functions for interfacing from Franz Lisp to the Unix operating system.

11.1. (time x_repeat x_mode g_expr)

Expression evaluation timing function. Defined as a fexpr (nlambda). The CPU time reported includes garbage collection time.

x_repeat:
Number of times Expr will be evaluated

x_mode:
0: CPU and REAL time
1: CPU time only
2: CPU and GC time
3: CPU, GC, and REAL time

g_expr:
The expression whose evaluation time is being monitored.

Notes: should be revised to print average times for Repeat values > 1 (as in Interlisp) Args to the g_expr passed to "time" must be declared **special** to compile correctly. When compiling programs using time use (declare (*fexpr time))

11.2. (unix)

This function initializes or reinitializes the Unix function call package. This function is invoked when the sysfn.l file is loaded, but certain errors encountered while using the Lisp system may make it necessary to reinitialize the Unix function call package manually.

The Unix function call package provides simple access to "all" Unix functions from inside Franz Lisp via a simple "exec" call (Section 4.3 of the Franz Lisp Manual). Arguments to the Unix function may be supplied on the same command line, e.g.

-> more myfile.

The call works by intercepting Unbound Variable errors while error processing, so atoms with unix-function names should be left unbound. Unix functions are identified by having a UNIX property = t on the atom with the appropriate name. For simple initialization, these are placed on the list UnixFunctions and set-up is done by invoking the function "(unix)".

The following names are recognized as Unix functions by this system :

cat cd cp emacs grep imprint jobs liszt ll ls lpr mkdir mail more mv ps pwd rm rmdir ruptime rwho uptime who w ww vi

References

(1) –, *Interlisp Reference Manual,* Warren Teitelman, ed., Xerox Palo Alto Research Center, California, 1978.

(2) Foderaro, John K., Sklower, Keith L., and Layer, Kevin, *The FRANZ LISP Manual,* The University of California, Berkeley, California, June 1983.

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-F-85-937	2.	3. Recipient's Accession No.

4. Title and Subtitle

Macros and Utilities for Franz Lisp

5. Report Date
February 1985

6.

7. Author(s)
Jeff Becker

8. Performing Organization Rept. No.

9. Performing Organization Name and Address

Department of Computer Science
University of Illinois
Urbana, IL 61801

10. Project/Task/Work Unit No.

11. Contract/Grant No.
NSF 84-06801
ONR N00014-82-K-0186

12. Sponsoring Organization Name and Address

National Science Foundation
Washington, DC

Office of Naval Research
Arlington, VA

13. Type of Report & Period Covered

14.

15. Supplementary Notes

16. Abstracts

This paper is a user's guide to set macros and utilities for use with the Franz Lisp programming language. These facilities aid porting programs from Interlisp to Franz Lisp, make the programming task easier, and improve readability of programs.

17. Key Words and Document Analysis. 17a. Descriptors

Programming Language
Macros

17b. Identifiers/Open-Ended Terms

Franz Lisp
Interlisp

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 17
	20. Security Class (This Page UNCLASSIFIED	22. Price