

## CHAPTER 14

### MISCELLANEOUS

- (SYSTEMTYPE) [Function]  
The SYSTEMTYPE function is intended to allow programmers to write system-dependent code. SYSTEMTYPE returns a litatom corresponding to the implementation of Interlisp: D (for Interlisp- D), TOPS-20, TENEX, JERICO, or VAX.  
In Interlisp- D (and Interlisp- 10), (SELECTQ (SYSTEMTYPE) ) expressions are expanded at compile time so that this is an effective way to perform conditional compilation.
- (USERNAME A FLG) [Function]  
If A= NIL, returns login directory name; if A= T, returns connected directory name; if A is a number, USERNAME returns the user name corresponding to that user number.  
The value is usually returned as a string. If FLG is a string ptr, it is smashed. If FLG is not a string pointer and is non-NIL, USERNAME returns the value as an atom.
- (STORAGE FLG GCFLG) [Function]  
Prints the amount of storage used for various data types. The exact printout is implementation- dependent. STORAGE returns NIL.  
In Interlisp- 10, the storage used by a particular type is only accurate immediately following a garbage collection of a related type. If GCFLG= T, STORAGE will perform the necessary garbage collections before printing its results. If FLG= T, includes storage used by and assigned to the system.
- (DISMISS MSEC SWAIT TIMER) [Function]  
In Interlisp- 10, dismisses the program for MSEC SWAIT milliseconds, during which time the program uses no CPU time. Can be aborted by control- D, control- E, or control- B.  
In Interlisp- D, dismisses the current process for MSEC SWAIT milliseconds, using the timer TIMER if given (see page 14.11).
- (APROPOS STRING ALLFLG) [Function]  
(Currently only in Interlisp- D) Prints information about all litatoms in the Interlisp system which contain the string STRING. APROPOS will print the argument lists of litatoms with function definitions, the values of litatoms with variable bindings, and the property names defined for litatoms with property lists. If ALLFLG is NIL, this scan does not include "system internal" litatoms; otherwise, all litatoms are scanned.

## Saving Interlisp State

(NEGATE *x*)

[Function]

Returns the negation of *x*. For example:

(NEGATE '(MEMBER *x* *y*)) => (NOT (MEMBER *x* *y*))

(NEGATE '(EQ *x* *y*)) => (NEQ *x* *y*)

(NEGATE '(AND *x* (NLISTP *x*))) => (OR (NULL *x*) (LISTP *x*))

The following two functions are useful writing programs that wish to reuse a scratch list to collect together some result (Both of these compile open):

(SCRATCHLIST *LST* *x*<sub>1</sub> *x*<sub>2</sub> ... *x*<sub>*N*</sub>)

[NLambda NoSpread Function]

SCRATCHLIST sets up a context in which the value of *LST* is used as a “scratch” list. The expressions *x*<sub>1</sub>, *x*<sub>2</sub>, ... *x*<sub>*N*</sub> are evaluated in turn. During the course of evaluation, any value passed to ADDTOSCRATCHLIST will be saved, reusing CONS cells from the value of *LST*. If the value of *LST* is not long enough, new CONS cells will be added onto its end. If the value of *LST* is NIL, the entire value of SCRATCHLIST will be “new” (i.e. no CONS cells will be reused).

(ADDTOSCRATCHLIST *VALUE*)

[Function]

For use under calls to SCRATCHLIST. *VALUE* is added on to the end of the value being collected by SCRATCHLIST. When SCRATCHLIST returns, its value is a list containing all of the things that ADDTOSCRATCHLIST has added.

### 14.1 SAVING INTERLISP STATE

(LOGOUT *FAST*)

[Function]

Stops Interlisp, and returns control to the operating system. From there, it is possible to continue Interlisp as of the LOGOUT. LOGOUT will not affect the state of open les.

In Interlisp-D, LOGOUT writes out all altered pages from real memory to the le Lisp.virtualmem. This usually takes about 30 seconds on the Xerox 1100. If *FAST* is non-NIL, Interlisp is stopped without updating Lisp.virtualmem. Note that it will not be possible to restart Interlisp from the point of the LOGOUT, and it may not be possible to restart it at all. Typing (LOGOUT *T*) is preferable to just booting the machine, because it also does other cleanup operations (closing network connections, etc.).

In Interlisp-10, if Interlisp was started as a subsidiary fork (see SUBSYS, page 22.21), control is returned to the higher fork.

The function SYSOUT saves the current state of the Interlisp virtual memory on a le. The le package (page 11.1) can be used to save particular function definitions and other arbitrary objects on les, but SYSOUT saves the *total* state of the system.

The le produced by SYSOUT (known as “a sysout le”, or simply “a sysout”) can be restarted from the operating system (by typing LISP SYSOUTFILE in Interlisp-D or RUN SYSOUTFILE in Interlisp-10). This

## MISCELLANEOUS

will restart Interlisp, and restore the virtual memory to the exact state that it had when the sysout le was made.

(SYSOUT FILE)

[Function]

Saves the current state of the Interlisp virtual memory on the le FILE, in a form that can be subsequently restarted. The current state of program execution is saved in the sysout le, so (PROGN (SYSOUT 'FOO) (PRINT 'HELLO)) will cause HELLO to be printed after the sysout le is restarted.

If FILE is non-NIL, the variable SYSOUTFILE is set to the body of FILE. If FILE is NIL, then the value of SYSOUTFILE instead. Therefore, (SYSOUT) will save the current state on the next higher version of a le with the same name as the previous SYSOUT. Also, if the extension for FILE is not specified, the value of SYSOUT.EXT is used. This is initially SYSOUT in Interlisp-D, SAV in Tenex Interlisp-10, and EXE in Tops-20 Interlisp-10.

SYSOUT sets SYSOUTDATE to (DATE), the time and date that the SYSOUT was performed.

If SYSOUT was not able to create the sysout le, because of disk or computer error, or because there was not enough space on the directory, SYSOUT returns NIL. Otherwise it returns the full le name of FILE.

Actually, SYSOUT "returns" twice; when the sysout le is rst created, and when it is subsequently restarted. In the latter case, SYSOUT returns the list (FILE . MAKESYSFILE), where FILE is the sysout le, and MAKESYSFILE is the original Interlisp makesys le (see MAKESYS, below). For example, (if (LISTP (SYSOUT 'FOO)) then (PRINT 'HELLO)) will cause HELLO to be printed when the sysout le is restarted, but not when SYSOUT is initially performed.

Note: SYSOUT does not save the state of any open les. WHENCLOSE (page 6.11) can be used to associate certain operations with open les so that when a SYSOUT is started up, these les will be reopened, and le pointers repositioned.

In Interlisp-10, a sysout le only contains the parts of the virtual memory that the user has changed. When the sysout le is restarted, the other pages are taken from the makesys le of the Interlisp system within which the sysout le was made (see MAKESYS, below). Therefore, whenever the Interlisp system is reassembled and/or reloaded, old sysout les are *not* compatible with the new system.

In Interlisp-D, a sysout le contains a copy of the entire allocated virtual memory, so it is very large. A normal sized sysout le contains about 4000 pages. Unlike in Interlisp-10, a sysout le is copied into the virtual memory when it is restarted, so it is perfectly permissible to overwrite a sysout le on top of the currently running sysout, for example, (SYSOUT '{DSK}FOO.SYSOUT;1) to overwrite FOO.SYSOUT on the local disk. Not only is this permissible, it is much faster than making a new sysout le (almost twice as fast, due to less disk overhead). Making a sysout le on the Xerox 1100 currently takes at least 5 minutes.

SYSOUT evaluates the expressions on BEFORESYSOUTFORMS before creating the sysout le. This variable initially includes expressions to: (1) Set the variables SYSOUTDATE and SYSOUTFILE as described above; (2) Default the sysout le name FILE according to the values of the variables SYSOUTFILE and SYSOUT.EXT, as described above; and (3) Perform any necessary operations on open les as specified by calls to WHENCLOSE (page 6.11).

## Saving Interlisp State

After a sysout le is restarted (but *not* when it is initially created), SYSOUT evaluates the expressions on AFTERSYSOUTFORMS. This initially includes expressions to: (1) Perform any necessary operations on previously-opened les as speci ed by calls to WHENCLOSE (page 6.11); (2) [Interlisp- 10 only] Reset the terminal line length with SETLINELENGTH (page 6.8); (3) [Interlisp- 10 only] Reset the terminal control characters using SETTERMCHARS (page 17.59) if the operating system has changed from Tenex to Tops-20 or vice versa; (4) Possibly print a message, as determined by the value of SYSOUTGAG (see below); and (5) Call SETINITIALS to reset the initials used for time-stamping (page 17.60).

SYSOUTGAG [Variable]

The value of SYSOUTGAG determines what is printed when a sysout le is restarted. If the value of SYSOUTGAG is a list, the list is evaluated, and no additional message is printed. This allows the user to print a message. If SYSOUTGAG is non-NIL and not a list, no message is printed. Finally, if SYSOUTGAG is NIL (its initial value), and the sysout le is being restarted by the same user that made the sysout originally, the user is greeted by printing the value of HERALDSTRING (see below) followed by a greeting message. If the SYSOUT le was made by a di erent user, a message is printed, warning that the user pro les may be di erent (see page 14.5);

(SYSIN FILE) [Function]

[Interlisp- 10 only] Restores the state of Interlisp from a sysout le. This is essentially the same as exiting Interlisp, and restarting a sysout le from the operating system executive. If SYSIN returns NIL, there was a problem in reading the le. If FILE was not found, generates a FILE NOT FOUND error.

(SYSOUTP FILE) [Function]

[Interlisp- 10 only] Returns the name of the original Interlisp makesys le (see MAKESYS, below) if FILE is a sysout le, otherwise NIL.

FILE may also be a JFN.

(MAKESYS FILE NAME) [Function]

Used to store a new Interlisp system on the “makesys le” FILE. Before this is done, the system is “initialized” by undoing the greet history, and clearing the display [Interlisp-D].

When the system is rst started up, a “herald” is printed identifying the system, typically “Interlisp-xx DATE ...”. If NAME is non-NIL, MAKESYS will use it instead of Interlisp-xx in the herald. MAKESYS sets HERALDSTRING to the herald string printed out.

MAKESYS also sets the variable MAKESYSDATE to (DATE), i.e. the time and date the system was made.

In Interlisp-D, MAKESYS is almost the same as SYSOUT, except that it does some cleaning-up operations (such as clearing the screen). In Interlisp- 10, however, MAKESYS is considerably di erent from SYSOUT, because it saves *all* of the pages in the Interlisp virtual memory, and allows the makesys le to be shared between multiple users.

The Interlisp- 10 system initially obtained by the user is shared; that is, all active users of Interlisp- 10 are actually using the same pages of memory. As a user adds to the system, private pages are added to his memory. Similarly, if the user changes anything in the original shared Interlisp- 10, for example, by advising a system function, a private copy of the changed page is created.

## MISCELLANEOUS

In addition to the swapping time saved by having several users accessing the same memory, the sharing mechanism permits a large saving in garbage collection time, since it is not necessary to garbage collect any data in the shared system, and thus Interlisp-10 does not need to chase from any pointers on shared pages during garbage collections.

This reduction in garbage collection time is possible because the shared system usually is not modified very much by the user. If the shared system is changed extensively, the savings in time will vanish, because once a page that was initially shared is made private, every pointer on it must be assumed active, because it may be pointed to by something in the shared system. Since every pointer on an initially shared but now private page can also point to *private* data, they must always be chased.

A user may create his own shared system with the function `MAKESYS`. If several people are using the same system, making the system be shared will result in a savings in swapping time. Similarly, if a system is large and seldom modified, making it be shared will result in a reduction of garbage collection time, and may therefore be worthwhile even if the system is only being used by one user.

One problem with using `MAKESYS` in Interlisp-10 is that it may protect large amounts of useless data from being garbage collected. For example, suppose that during the course of building an Interlisp system, a large number of list cells are used and discarded. If `MAKESYS` is now executed to store the system, all of that list cell space is stored, and protected from garbage collection (unless the user changes those pages, making a personal copy). To solve this problem, it is necessary to make sure that as little storage as possible is allocated while creating a new system, perhaps by setting `MINFS` (page 22.10) to a very low value. Of course, this will slow down Interlisp considerably, so making a new system will take a long time.

### 14.2 GREETING AND USER PROFILES

Many of the features of Interlisp are parameterized to allow the user to adjust the system to his or her own tastes. Among the more commonly adjusted parameters are `PROMPT#FLG` (page 8.18), `DWIMWAIT` (page 15.11), `CHANGESLICE` (page 8.18), `LOWERCASE` (page 16.21), `#UNDOSAVES` (page 8.33), `INITIALSLST` (page 17.60), etc. In addition, the user can modify the action of system functions in ways not specially provided for by using `ADVISE` (page 10.9).

In order to encourage this procedure, and to make it as painless and automatic as possible, the programmer's assistant includes a facility for both a site-defined profile and a user-defined profile. When Interlisp is first run, it calls the function `GREET` (see below). This provides a way of setting defaults for a particular community of users, patching bugs, etc.

Greeting (i.e., the initialization) is undoable, and is stored as a separate event on the history list (page 8.25). The user can explicitly invoke the greeting operation at any time via the function `GREET`. This can also be used to effect another user's initialization.

(`GREET` `NAME` `_`)

[Function]

Performs the greeting for the user whose username is `NAME` (if `NAME` = `NIL`, uses the login name). When Interlisp first starts up, it performs (`GREET`).

Before `GREET` performs the indicated initialization, it first undoes the effects of the previous greeting. The side effects of the greeting operation are stored on a global variable as well as the history list, thus enabling the previous greeting to be undone

## Manipulating File Directories

even if it is no longer on the history list. In addition, MAKESYS is advised to undo the effects of the previous greeting, thereby returning the system to a pristine state.

GREET initializes in the following way: It first evaluates each item in the list PREGREETFORMS, then it loads the file returned from (GREETFILENAME T), then it loads the file returned from (GREETFILENAME USERNAME), then it evaluates each item on POSTGREETFORMS, and finally it prints a greeting such as 'Hello, xxx .', where xxx is the FIRSTNAME component of the user's entry on INITIALSLIST (page 17.60). The loads are performed "silently" by rebinding PRETTYHEADER (page 11.36) to NIL.

(GREETFILENAME USER) [Function]  
GREETFILENAME is a system-dependent function. Its purpose is to locate existing files used for greeting and return them. If USER is T, then it returns the filename of the site-defined profile (if it exists). Otherwise, USER is interpreted to be a user's system name, and it returns the filename for the user-defined profile (if it exists).

GREETDATES [Variable]  
The value of GREETDATES can be used to specify special greeting messages for various dates. GREETDATES is a list of elements of the form (DATESTRING . STRING), e.g. ("25-DEC" . "Merry Christmas"). The user can add entries to this list in his/her INIT.LISP file by using a ADDVARS file package command like (ADDVARS (GREETDATES ("8-FEB" . "Happy Birthday"))). On the specified date, the GREET will use the indicated salutation.

### 14.3 MANIPULATING FILE DIRECTORIES

The following function allows the user to conveniently specify and/or program a variety of directory operations:

(DIRECTORY FILES COMMANDS DEFAULTTEXT DEFAULTTVERS) [Function]  
FILES is either [1] NIL (which is equivalent to \*.\*); or [2] an atom which can contain \$'s or \*'s (equivalent) which match any number of characters or ?'s which match a single character, or else [3] FILES is a list of the form (FILES + FILES), (FILES - FILES), or (FILES \* FILES),<sup>1</sup> e.g., (T\$ + \$L) will match with any file beginning with T or ending in L, (T\$ - \*.DCOM) matches all files that begin with T and are not .DCOM files.

For each file that matches, each command in COMMANDS is executed with the following interpretation:

P	Print file name.
PP	Print file name (except for version number).
a string	Prints the string.

---

<sup>1</sup>OR can be used for +, and AND for \*.

## MISCELLANEOUS

READDATE, WRITEDATE, CREATIONDATE  
 SIZE, LENGTH, BYTESIZE  
 PROTECTION, AUTHOR, TYPE

Prints the appropriate information returned by GETFILEINFO (page 6.6).

COLLECT            The value of DIRECTORY will be a list of le names; add the complete le name of this le to that list.

COUNTSIZE        The value of DIRECTORY will be a sum; add the size of this le to that sum.

PAUSE            Wait until the user types any char before proceeding with the rest of the commands (good for display if you want to ponder).

PROMPT MESS      Prompts with MESS ; if user responds with No, abort command processing for this le.

OLDERTHAN N      Continue command processing if the le hasn't been referenced (read or written) in N days.

OLDVERSIONS N    Continue command processing if there are at least N more recent versions of the same le.

BY USER          Continue command processing if the le was last written by the given user.

@ X              X is either a function of one argument (JFN), a function of two arguments (JFN FILENAME ) or an arbitrary expression which uses the variables JFN and/or the variables FILENAME freely. If X returns NIL, abort command processing for this le.

DELETED          Allows DIRECTORY to examine deleted les (normally, they are not mapped over.

OUT FILE         Directs output to FILE.

COLUMNS N      Attempt to format output in N columns (rather than just 1).

TRIMTO N         Deletes all but N versions of le (N 0).

DELETE           Deletes le. If this is speci ed, the value of DIRECTORY is NIL if no COLLECT command is speci ed, otherwise the list of les deleted.

UNDELETE         Undeletes the indicated les that have been deleted.

DIRECTORY uses DIRCOMMANDS to correct spelling, which also provides a way of de ning abbreviations and synonyms (page 15.13). Currently the following abbreviations are recognized:

AU               =>     AUTHOR

-                =>     PAUSE

COLLECT?       =>     PROMPT " ? " COLLECT

DA

TI               =>     WRITEDATE

## Sorting Lists

DEL => DELETE

DEL?

DELETE? => PROMPT " delete? " DELETE

OLD => OLDERTHAN 90

PR => PROTECTION

SI => SIZE

(FILDIR FILEGR OUP \_ ) [Function]  
 FILEGR OUP is a le group descriptor, i.e., it can contain stars. FILDIR returns a list of the les which match FILEGR OUP , a la the DIRECTORY function, e.g., (FILDIR '\*.COM;0).

There is also a programmer's assistant command DIR which calls the function DIRECTORY:

DIR FILES . COMMANDS [Prog. Asst. Command]  
 Calls the function DIRECTORY with (P . COMMANDS ) as the command list and \* and \* as the default extension and default version respectively.

For example, to DELVER only those les which you ok, do DIR FILES PROMPT "?" TRIMTO 1.

## 14.4 SORTING LISTS

(SORT DATA COMP AREFN ) [Function]  
 DATA is a list of items to be sorted using COMP AREFN , a predicate function of two arguments which can compare any two items on DATA and return T if the rst one belongs before the second. If COMP AREFN is NIL, ALPHORDER is used; thus (SORT DATA) will alphabetize a list. If COMP AREFN is T, CAR's of items that are lists are given to ALPHORDER, otherwise the items themselves; thus (SORT A-LIST T) will alphabetize an assoc list by the CAR of each item. (SORT X 'ILESSP) will sort a list of integers.

The value of SORT is the sorted list. The sort is destructive and uses no extra storage. The value returned is EQ to DATA but elements have been switched around. Interrupting with control D, E, or B may cause loss of data, but control H may be used at any time, and SORT will break at a clean state from which ^ or control characters are safe. The algorithm used by SORT is such that the maximum number of compares is  $N \cdot \log_2 N$ , where N is (LENGTH DATA).

Note: if (COMP AREFN A B) = (COMP AREFN B A), then the ordering of A and B may or may not be preserved.

For example, if (FOO . FIE) appears before (FOO . FUM) in X, (SORT X T) may or may not reverse the order of these two elements. Of course, the user can always specify a more precise COMP AREFN .



## MISCELLANEOUS

(MERGE A B COMP AREFN ) [Function]  
A and B are lists which have previously been sorted using SORT and COMP AREFN . Value is a destructive merging of the two lists. It does not matter which list is longer. After merging both A and B are equal to the merged list. (In fact, (CDR A) is EQ to (CDR B)). MERGE may be aborted after control-H.

(ALPHORDER A B) [Function]  
A predicate function of two arguments, for alphabetizing. Returns T if its arguments are in order, i.e., if B does not belong before A. Numbers come before literal atoms, and are ordered by magnitude (using GREATERP). Literal atoms and strings are ordered by comparing the character codes in their pnames. Thus (ALPHORDER 23 123) is T, whereas (ALPHORDER 'A23 'A123) is NIL, because the character code for the digit 2 is greater than the code for 1.

Atoms and strings are ordered before all other data types. If neither A nor B are atoms or strings, the value of ALPHORDER is T, i.e., in order.

Note: ALPHORDER does no UNPACKs, CHCONS, CONSES or NTHCHARs. It is several times faster for alphabetizing than anything that can be written using these other functions.

(MERGEINSERT NEW LST ONEFL G) [Function]  
LST is NIL or a list of partially sorted items. MERGEINSERT tries to find the "best" place to (destructively) insert NEW , e.g.,  
  
(MERGEINSERT 'FIE2 '(FOO FOOL FIE FUM))  
=> (FOO FOOL FIE FIE2 FUM)

Returns LST. MERGEINSERT is undoable.

If ONEFL G = T and NEW is already a member of LST, MERGEINSERT does nothing and returns LST.

MERGEINSERT is used by ADDTOFILE (page 11.33) to insert the name of a new function into a list of functions. The algorithm is essentially to look for the item with the longest common leading sequence of characters with respect to NEW , and then merge NEW in starting at that point.

(COMPARELISTS X Y) [Function]  
Compares X and Y and prints their differences, i.e., COMPARELISTS is essentially a SRCCOM for list structures.

## 14.5 DATE/TIME FUNCTIONS

(DATE \_ ) [Function]  
Obtains date and time, returning it as a single string with format "DD-MM-YY HH:MM:SS", where DD is day, MM is month, YY year, HH hours, MM minutes, SS seconds, e.g., "14-MAY-71 14:26:08".

In Interlisp-10, DATE will accept FORMA TBITS as an argument, which can be used

## Timers and Duration Functions

to specify other formats, e.g., day of week, time zone, etc., as described in the JSYS manual.

(IDATE STR) [Function]  
 STR is a date and time string. Value of IDATE is STR converted to a number such that if DATE<sub>1</sub> is before (earlier than) DATE<sub>2</sub>, then (IDATE DATE<sub>1</sub>) < (IDATE DATE<sub>2</sub>). (IDATE) returns (IDATE (DATE)).

(GDATE DATE FORMA TBITS STRPTR) [Function]  
 Interlisp-10 function for obtaining time-date formatted string, DATE is in internal date-and-time format. If NIL, current time and date is used, i.e. value of (IDATE). FORMA TBITS is 36 bit quantity to be passed to TENEX/TOPS 20 time-date conversion routines (see JSYS manual). For example, FORMA TBITS=-1 gives a "long" date, e.g. "FRIDAY, JUN 16, 1978, 23:41:52-PDT". If FORMA TBITS= NIL, defaults to a value which will produce the same format as that of (DATE), i.e. "DD-MM-YY HH:MM:SS". STRPTR is an optional string pointer to be reused. In this case, the string characters are stored in an internal scratch string, MACSCRATCHSTRING, so that a subsequent call to GDATE will overwrite the characters returned by this one. Note that this internal scratch string is also used by several other functions in this section.

The dateformat package (page 23.57) provides a convenient way of specifying the format bits in terms of keywords.

(CLOCK N) [Function]  
 For N=0, returns the current value of the time of day clock i.e., number of milliseconds since last system start up.  
 For N=1, returns the value of the time of day clock when the user started up this Interlisp, i.e., difference between (CLOCK 0) and (CLOCK 1) is number of milliseconds (real time) since this Interlisp was started.  
 For N=2, returns the number of milliseconds of *compute* time since user started up this Interlisp (garbage collection time is subtracted off).  
 For N=3, returns the number of milliseconds of compute time spent in garbage collections (all types).<sup>2</sup>

## 14.6 TIMERS AND DURATION FUNCTIONS

Often one needs to loop over some code, stopping when a certain interval of time has passed. Some systems provide an "alarmclock" facility, which provides an asynchronous interrupt when a time interval runs out. This is not particularly feasible in the current Interlisp-D environment, so the following facilities are supplied for efficiently testing for the expiration of a time interval in a loop context.

---

<sup>2</sup>In Interlisp-10, this number is directly accessible via the COREVAL GCTIM.

## MISCELLANEOUS

Three functions are provided: `SETUPTIMER`, `SETUPTIMER.DATE`, and `TIMEREXPIRED?`. Also several new i.s.oprs have been dened: `forDuration`, `during`, `untilDate`, `timerUnits`, `usingTimer`, and `resourceName` (reasonable variations on upper/lower case are permissible).

These functions use an object called a `Timer`, which encodes a future clock time at which a signal is desired. A `Timer` is constructed by the functions `SETUPTIMER` and `SETUPTIMER.DATE`, and is created with a basic clock “unit” selected from among `SECONDS`, `MILLISECONDS`, or `TICKS`. The rst two timer units provide a machine/system independent interface, and the latter provides access to the “real”, basic strobe unit of the machine’s clock on which the program is running. The default unit is `MILLISECONDS`.

Currently, the `TICKS` unit is the same as the `MILLISECONDS` unit for `Interlisp-10` and `Interlisp/VAX`. In `Interlisp-D`, the `TICKS` unit is a function of the particular machine that `Interlisp-D` is running on: The `Xerox 1100` and `1132` have about 0.5952 microseconds per tick (1680 ticks per millisecond); The `Xerox 1108` has about 28.78 microseconds per tick (34.746 ticks per millisecond). The advantage of using `TICKS` rather than one of the uniform interfaces is primarily speed; e.g., on a `Xerox 1100`, it may take as much as 400 microseconds to interface the milliseconds clock (a software facility actually based over the real clock), whereas reading the real clock itself should take less than about ten microseconds. The disadvantage of the `TICKS` unit is its short roll-over interval (about 20 minutes) compared to the `MILLISECONDS` roll-over interval (about about two weeks), and also the dependency on particular machine parameters.

(`SETUPTIMER` `INTERVAL` `OLDTIMER?` `TIMER UNITS` `INTERVAL UNITS`) [Function]  
`SETUPTIMER` returns a `Timer` that will “go o” (as tested by `TIMEREXPIRED?`) after a speci ed time-interval measured from the current clock time. `SETUPTIMER` has one required and three optional arguments:

`INTERVAL` must be a integer specifying how long an interval is desired. `TIMER UNITS` speci es the units of measure for the interval (defaults to `MILLISECONDS`).

If `OLDTIMER?` is a `Timer`, it will be reused and returned, rather than allocating a new `Timer`. `INTERVAL UNITS` speci es the units in which the `OLDTIMER?` is expressed (defaults to the value of `TIMER UNITS`).

(`SETUPTIMER.DATE` `DTS` `OLDTIMER?`) [Function]  
`SETUPTIMER.DATE` returns a `Timer` (using the `SECONDS` time unit) that will “go o” at a speci ed date and time. `DTS` is a `Date/Time` string such as `IDATE` accepts (page 14.10). If `OLDTIMER?` is a `Timer`, it will be reused and returned, rather than allocating a new `Timer`.

`SETUPTIMER.DATE` operates by rst subtracting (`IDATE`) from (`IDATE DTS`), so there may be some large integer creation involved, even if `OLDTIMER?` is given.

(`TIMEREXPIRED?` `TIMER` `CLOCKVALUE.OR.TIMER UNITS`) [Function]  
If `TIMER` is a `Timer`, and `CLOCKVALUE.OR.TIMER UNITS` is the time-unit of `TIMER`, `TIMEREXPIRED?` returns true if `TIMER` has “gone o”.

`CLOCKVALUE.OR.TIMER UNITS` can also be a `Timer`, in which case `TIMEREXPIRED?` compares the two timers (using the same time units). If `X` and `Y` are `Timers`, then (`TIMEREXPIRED? X Y`) is true if `X` is set for a *later* time than `Y`.

There are a number of i.s.oprs that make it easier to use `Timers` in iterative statements (page 4.5). These i.s.oprs are given below in the “canonical” form, with the second “word” capitalized, but the all-caps and all-lower-case versions are also acceptable.

## Timers and Duration Functions

`forDuration` `INTER VAL` [I.S. Operator]  
`during` `INTER VAL` [I.S. Operator]  
    `INTER VAL` is an integer specifying an interval of time during which the iterative statement will loop.

`timerUnits` `UNITS` [I.S. Operator]  
    `UNITS` species the time units of the `INTER VAL` specied in `forDuration`.

`untilDate` `DTS` [I.S. Operator]  
    `DTS` is a Date/Time string (such as `IDATE` accepts) specifying when the iterative statement should stop looping.

`usingTimer` `TIMER` [I.S. Operator]  
    If `usingTimer` is given, `TIMER` is reused as the timer for `forDuration` or `untilDate`, rather than creating a new timer. This can reduce allocation if one of these i.s.oprs is used within another loop.

`resourceName` `RESOUR CE` [I.S. Operator]  
    `RESOUR CE` species a `GLOBALRESOURCES` name to be used as the timer storage. If `RESOUR CE = T`, it will be converted to a common internal name.

Some examples:

```
(during 6MONTHS timerUnits 'SECS
until (TENANT-VACATED? HouseHolder)
do (DISMISS <for-about-a-day>)
  (HARRASS HouseHolder)
finally (if (NOT (TENANT-VACATED? HouseHolder))
  then (EVICT-TENANT HouseHolder)))
```

This humorous little example shows that how is is possible to have two termination condition: (1) when the time interval of 6MONTHS has elapsed, or (2) when the predicate `(TENANT-VACATED? HouseHolder)` becomes true. Note that the “nally” clause is executed regardless of which termination condition caused it.

```
(do (forDuration (CONSTANT (ITIMES 10 24 60 60 1000))
  do (CARRY.ON.AS.USUAL)
  finally (PROMPTPRINT "Have you had your 10-day check-up?")))
```

This in nite loop breaks out with a warning message every 10 days. One could question whether the millisecond clock, which is used by default, is appropriate for this loop, since it rolls-over about every two weeks.

```
(SETQ \RandomTimer (SETUPTIMER 0))
(untilDate "31-DEC-83 23:59:59" usingTimer \RandomTimer
when (WINNING?) do (RETURN)
finally (ERROR "You've been losing this whole year!"))
```

Here we see a usage of an explicit date for the time interval; also, the user has squirreled away some storage (as the value of `\RandomTimer`) for use by the call to `SETUPTIMER` in this loop.

```
(forDuration SOMEINTERVAL
resourceName '\INNERLOOPBOX
```

## MISCELLANEOUS

```
timerunits 'TICKS
do (CRITICAL.INNER.LOOP))
```

For this loop, the user doesn't want any CONSing to take place, so \INNERLOOPBOX will be defined as a GLOBALRESOURCE which "caches" a timer cell (if it isn't already so defined), and wraps the entire statement in a GLOBALRESOURCE call. Furthermore, he has specified a time unit of TICKS, for lower overhead in this critical inner loop. In fact specifying a resource name of T would have been the same as specifying it to be \ForDurationOfBox; this is just a simpler way to specify that a GLOBALRESOURCE is wanted, without having to think up a name.

### 14.7 GAINSPACE

For users with large programs and data bases, the user may sometimes find himself in a situation where he needs to obtain more space, and is willing to pay the price of eliminating some or all of the context information that the various user-assistance facilities such as the programmer's assistant, le package, CLISP, etc., have accumulated during the course of his session. The following function is available for this purpose.

```
(GAINSPACE) [Function]
Prints a list of deletable objects, allowing the user to specify at each point what
should be discarded and what should be retained.
```

For example:

```
_ (GAINSPACE)
purge history lists ? Yes
purge everything, or just the properties, e.g., SIDE, LISPXPRINT, etc. ?
just the properties
discard definitions on property lists ? Yes
discard old values of variables ? Yes
erase properties ? No
erase CLISP translations? Yes
.
.
.
```

GAINSPACE is driven by the list GAINSPACEFORMS. Each element on GAINSPACEFORMS is of the form (PRECHECK MESSAGE FORM KEYLST). If PRECHECK, when evaluated, returns NIL, GAINSPACE skips to the next entry. For example, the user will not be asked whether or not to purge the history list if it is not enabled. Otherwise, ASKUSER (page 6.57) is called with the indicated MESSAGE and the (optional) KEYLST. If the user responds No, i.e., ASKUSER returns N, GAINSPACE skips to the next entry. Otherwise, FORM is evaluated with the variable RESPONSE bound to the value of ASKUSER. In the above example, the FORM for the "purge history lists" question calls ASKUSER to ask "purge everything, " only if the user had responded Yes. If the user had responded with Everything, the second question would not have been asked.

The "erase properties" question is driven by a list SMASHPROPSMENU. Each element on this list is of the form (MESSAGE . PROPS). The user is prompted with MESSAGE (by ASKUSER), and if he

## Performance Measuring Functions

responds Yes, `PROPS` is added to the list `SMASHPROPS`. The ‘discard definitions on property lists’ and ‘discard old values of variables’ questions also add to `SMASHPROPS`. The user will not be prompted for any entry on `SMASHPROPSMENU` for which all of the corresponding properties are already on `SMASHPROPS`. `SMASHPROPS` is initially set to the value of `SMASHPROPSLIST`. This permits the user to specify in advance those properties which he always wants to be discarded, and not be asked about them subsequently. After nishing all the entries on `GAINSPACEFORMS`, `GAINSPACE` checks to see if the value of `SMASHPROPS` is non-NIL, and if so, does a `MAPATOMS`, i.e., looks at every atom in the system, and erases the indicated properties.

Note that the user can change or add new entries to `GAINSPACEFORMS` or `SMASHPROPSMENU`, so that `GAINSPACE` can also be used to purge structures that the user’s programs have accumulated.

### 14.8 PERFORMANCE MEASURING FUNCTIONS

(CONSCOUNT *N*) [Function]  
(CONSCOUNT) returns the number of CONSES since Interlisp started up. If *N* is not NIL, resets CONSCOUNT to *N*.

(BOXCOUNT *TYPE N*) [Function]  
Returns the number of boxing operations for the data type *TYPE* (see page 2.36) since Interlisp started up. If *N* is not NIL, the corresponding counter is reset to *N*.  
  
In Interlisp-10, if *TYPE* = NIL, BOXCOUNT returns the number of large integer boxes; if *TYPE* is non-NIL, it returns the number of floating boxes. These counters are directly accessible via the `COREVALS` `IBOXCN` and `FBOXCN`.

In Interlisp-D, *TYPE* can be any datatype name, in addition to `FIXP` and `FLOATP`.

(PAGEFAULTS) [Function]  
Returns the number of page faults since Interlisp started up.

(TIME *TIMEX TIMEN TIMETYPE*) [NLambda Function]  
An nlambda function. It executes the computation *TIMEX*, and prints out the number of conses and computation time. Garbage collection time is subtracted out. For example, in Interlisp-10:

```
_TIME((LOAD (QUOTE PRETTY) (QUOTE PROP])
FILE CREATED 1-AUG-78 14:56:12
PRETTYCOMS
collecting lists
582, 10291 free cells
13169 CONSES
29.484 SECONDS
PRETTY
```

If *TIMEN* is greater than 1 (*TIMEN* = NIL is equivalent to *TIMEN* = 1), *TIMEX* is executed *TIMEN* times, and *TIME* prints out (number of conses)/*TIMEN*, and (computation time)/*TIMEN*. This is useful for more accurate measurement on small computations, e.g.

## MISCELLANEOUS

```
_TIME((COPY (QUOTE (A B C))) 10)
30/10 = 3 CONSES
.055/10 = .0055 SECONDS
(A B C)
```

If `TIMETYPE` is 0, `TIME` measures and prints total *real* time as well as computation time, e.g.

```
_TIME((LOAD (QUOTE PRETTY) (QUOTE PROP)) 1 0]
FILE CREATED 7-MAY-71 12:47:14
GC: 8
582, 10291 FREE WORDS
PRETTYFNS
PRETTYVARS
3727 CONSES
11.193 SECONDS
27.378 SECONDS, REAL TIME
PRETTY
```

If `TIMETYPE` = 3, `TIME` measures and prints garbage collection time as well as computation time, e.g.

```
_TIME((LOAD (QUOTE PRETTY) (QUOTE PROP)) 1 3]
FILE CREATED 7-MAY-71 12:47:14
GC: 8
582, 1091 FREE WORDS
PRETTYFNS
PRETTYVARS
3727 CONSES
10.597 SECONDS
1.487 SECONDS, GARBAGE COLLECTION TIME
PRETTY
```

Another option is `TIMETYPE` = T, in which case `TIME` measures and prints the number of pagefaults.

The value of `TIME` is the value of the last evaluation of `TIMEX`.

### 14.8.1 BREAKDOWN

`TIME` collects statistics for whole computations. `BREAKDOWN` is available to analyze the breakdown of computation time (or any other measureable quantity) function by function.

(BREAKDOWN FN<sub>1</sub> FN<sub>N</sub>) [NLambda NoSpread Function]  
The user calls `BREAKDOWN` giving it a list of function names (unevaluated). These functions are modified so that they keep track of various statistics.

To remove functions from those being monitored, simply `UNBREAK` (page 10.6) the functions, thereby restoring them to their original state. To add functions, call `BREAKDOWN` on the new functions. This will not reset the counters for any functions not on the new list. However (BREAKDOWN) will zero the counters of

## BREAKDOWN

all functions being monitored.

The procedure used for measuring is such that if one function calls other and both are “broken down”, then the time (or whatever quantity is being measured) spent in the inner function is *not* charged to the outer function as well.

Note: BREAKDOWN will *not* give accurate results if a function being measured is not returned from normally, e.g., a lower RETFROM (or ERROR) bypasses it. In this case, all of the time (or whatever quantity is being measured) between the time that function is entered and the time the next function being measured is entered will be charged to the first function.

(BRKDOWNRESULTS RETURNV ALUESFLAG) [Function]  
BRKDOWNRESULTS prints the analysis of the statistics requested as well as the number of calls to each function. If RETURNV ALUESFLAG is non-NIL, BRKDOWNRESULTS will not print the results, but instead return them in the form of a list of elements of the form (FNNAME #CALLS VALUE).

Example:

```
_ (BREAKDOWN SUPERPRINT SUBPRINT COMMENT1)
(SUPERPRINT SUBPRINT COMMENT1)
_ (PRETTYDEF '(SUPERPRINT) 'FOO)
FOO.;3
_ (BRKDOWNRESULTS)
FUNCTIONS      TIME      #CALLS    PER CALL    %
SUPERPRINT     8.261      365      0.023      20
SUBPRINT       31.910     141      0.226      76
COMMENT1       1.612       8       0.201       4
TOTAL          41.783     514      0.081
NIL
_ (BRKDOWNRESULTS T)
((SUPERPRINT 365 8261) (SUBPRINT 141 31910) (COMMENT1 8 1612))
```

BREAKDOWN can be used to measure other statistics, by setting the following variables:

BRKDWNTYPE [Variable]

To use BREAKDOWN to measure other statistics, before calling BREAKDOWN, set the variable BRKDWNTYPE to the quantity of interest, e.g., TIME, CONSES, etc, or a list of such quantities. Whenever BREAKDOWN is called with BRKDWNTYPE not NIL, BREAKDOWN performs the necessary changes to its internal state to conform to the new analysis. In particular, if this is the first time an analysis is being run with a particular statistic, a measuring function will be defined, and the compiler will be called to compile it. The functions being broken down will be redefined to call this measuring function. When BREAKDOWN is through initializing, it sets BRKDWNTYPE back to NIL. Subsequent calls to BREAKDOWN will measure the new statistic until BRKDWNTYPE is again set and a new BREAKDOWN performed.

BRKDWNTYPES [Variable]

The list BRKDWNTYPES contains the information used to analyze new statistics. Each entry on BRKDWNTYPES should be of the form (TYPE FORM FUNCTION), where TYPE is a statistic name (as would appear in BRKDWNTYPE), FORM



## MISCELLANEOUS

computes the statistic, and `FUNCTION` (optional) converts the value of form to some more interesting quantity. For example, `(TIME (CLOCK 2) (LAMBDA (X) (FQUOTIENT X 1000)))` measures computation time and reports the result in seconds instead of milliseconds. `BRKDWNTYPES` currently contains entries for `TIME`, `CONSES`, `PAGEFAULTS`, `BOXES`, and `FBOXES`.

Example:

```
_ (SETQ BRKDWNTYPE '(TIME CONSES))
(TIME CONSES)
_ (BREAKDOWN MATCH CONSTRUCT)
(MATCH CONSTRUCT)
_ (FLIP '(A B C D E F G H C Z) '(... $1 .. #2 ...) '(... #3 ...))
(A B D E F G H Z)
_ (BRKDOWNRESULTS)
FUNCTIONS      TIME      #CALLS    PER CALL    %
MATCH          0.036      1         0.036      54
CONSTRUCT      0.031      1         0.031      46
TOTAL          0.067      2         0.033
FUNCTIONS      CONSES     #CALLS    PER CALL    %
MATCH          32         1         32.000     40
CONSTRUCT      49         1         49.000     60
TOTAL          81         2         40.500
NIL
```

Occasionally, a function being analyzed is sufficiently fast that the overhead involved in measuring it obscures the actual time spent in the function. If the user were using `TIME`, he would specify a value for `TIMEN` greater than 1 to give greater accuracy. A similar option is available for `BREAKDOWN`. The user can specify that a function(s) be executed a multiple number of times for each measurement, and the average value reported, by including a number in the list of functions given to `BREAKDOWN`, e.g., `BREAKDOWN(EDITCOM EDIT4F 10 EDIT4E EQP)` means normal breakdown for `EDITCOM` and `EDIT4F` but executes (the body of) `EDIT4E` and `EQP` 10 times each time they are called. Of course, the functions so measured must not cause any harmful side effects, since they are executed more than once for each call. The printout from `BRKDOWNRESULTS` will look the same as though each function were run only once, except that the measurement will be more accurate.

Another way of obtaining more accurate measurement is to expand the call to the measuring function in-line. If the value of `BRKDWNCOMPFLG` is non-NIL (initially NIL), then whenever a function is broken-down, it will be redefined to call the measuring function, and then recompiled. The measuring function is expanded in-line via an appropriate macro. In addition, whenever `BRKDWNTYPE` is reset, the compiler is called for *all* functions for which `BRKDWNCOMPFLG` was set at the time they were originally broken-down, i.e. the setting of the flag at the time a function is broken-down determines whether the call to the measuring code is compiled in-line.

### 14.9 PAGE MAPPED FILES

This facility allows paged access to files. It manages a set of paging buffers as a least-recently-used queue, with each buffer being a full-page block. Facilities are provided for allocating and deallocating buffers,

## Page Mapped Files

locking down pages, mapping a given page of the file into core, and getting the in-core location to which a given word of the file has been mapped. Any number of files can be mapped in at one time.

Note: Interlisp-D implements the page-mapping primitives of Interlisp-10 with some notable differences that might require major reworking of programs that rely on these facilities. The major difference is that an Interlisp-D page contains 256 16-bit words, rather than the 512 36-bit words of Interlisp-10. A given page number or file address for MAPPAGE or MAPWORD will correspond to a very different number of bits from the beginning of the file, and WORDCONTENTS and SETWORDCONTENTS move smaller amounts of information. A second difference is that buffers are completely integrated into the Interlisp-D storage management system so that a page is guaranteed to be locked down as long as the user holds a pointer to it. The functions LOCKMAP and UNLOCKMAP are therefore unnecessary, but for compatibility are defined with dummy definitions.

The following scenario illustrates the use of these facilities: The user first opens the file (or files) that he wants to access by page-mapping using any of the ordinary file-opening functions. Then, to examine a particular word in one of the files, the user simply gives the word number and the file's name to the function MAPWORD, which returns a pointer to the in-core location that that word is mapped to (i.e. the address as an unboxed number). When he has finished processing, the user simply closes the file (e.g. using CLOSEF) and the buffers are automatically unmapped.

The basic functions are:

(ADDMAPBUFFER TEMP ERRORFLAG) [Function]

Initially, a single buffer is allocated, so that page-mapping may be done without further initialization. More buffers can be allocated by ADDMAPBUFFER, which may help to avoid thrashing. ADDMAPBUFFER attempts to allocate a single new buffer, and returns non-NIL if successful. If there is not enough space to allocate a new buffer, then if ERRORFLAG is NIL, ADDMAPBUFFER simply returns NIL. Otherwise, ADDMAPBUFFER causes an error UNABLE TO ALLOCATE PMAP BUFFER.

If TEMP = T, the buffers are allocated on a "temporary" basis: allocation takes place via a RESETSAVE whose restoration form will de-allocate the buffers.

(MAPBUFFERCOUNT ONLYUNLOCKED) [Function]

Returns the number of buffers currently allocated. If ONLYUNLOCKED = T, counts only unlocked buffers; otherwise, counts all buffers. Thus, to insure that at least 3 (unlocked) buffers are allocated, the user could perform (while (LESSP (MAPBUFFERCOUNT T) 3) do (ADDMAPBUFFER NIL T)).

(MAPPAGE PAGEQ FILE \_) [Function]

The primitive function for mapping in pages from FILE into the queue of buffers. PAGEQ is a page number in FILE. The value of MAPPAGE is a pointer to the word in memory at which the first word of the page is located, which will always be at a page-boundary.

If FILE is NIL, the value of DEFAULTMAPFILE is used.

MAPPAGE searches the buffers to see if the given page for the given file has already been mapped in. If so, it returns the core address to which it was previously mapped. Otherwise, it replaces the previous contents of the least-recently-used buffer with the specified file page. It is important to note that the contents of a given core buffer are not guaranteed across calls to MAPPAGE, unless the page has

## MISCELLANEOUS

been locked down via LOCKMAP. MAPPAGE compiles open, and in the case where the desired page is already in the bufer it is quite efficient.

MAPPAGE will allocate an additional bufer if no unlocked buffers are available (and the desired page is not already mapped in).

In Interlisp-10, FILE may also be a fork handle (i.e. a value of SUBSYS, page 22.21), in which case the specified page from that fork will be mapped in.

(MAPWORD FILEADR FILE) [Function]  
Like MAPPAGE, except that it allows the specification of a word-address in FILE, not just a page number. MAPWORD determines what page that address is on, maps that page into a bufer (using MAPPAGE), and returns a pointer into the middle of the bufer where the indicated word appears. The rest of the words on the same file page appear at the appropriate word offsets from the value returned by MAPWORD.

(WORDOFFSET PTR N) [Function]  
If PTR is a pointer into a bufer as returned by MAPPAGE or MAPWORD, WORDOFFSET returns a pointer to the Nth following word. WORDOFFSET compiles open.

(WORDCONTENTS PTR) [Function]  
Returns the contents of the word at PTR as an integer. For example, (WORDCONTENTS (MAPWORD 10 FILE)) will return the value stored in word 10 of a (binary) file. WORDCONTENTS compiles open.

(SETWORDCONTENTS PTR N) [Function]  
Sets the contents of the word pointed to by PTR to be the number N. Interpreted, SETWORDCONTENTS checks that PTR actually is a pointer as returned by MAPPAGE or MAPWORD. SETWORDCONTENTS compiles open with no error checks.

(CLEARMAP FILE PAGES RELEASE) [Function]  
FILE specifies a file or fork as for MAPPAGE, or it is T. PAGES is a single page number or a list of page numbers. CLEARMAP unmaps any of those pages that are currently mapped in, making those buffers available for other mappings. FILE = T means all files; PAGES = NIL means all pages. Thus (CLEARMAP T) will completely clear the buffers.

Note that CLEARMAP unmaps any pages, whether or not they are currently locked, i.e., CLEARMAP takes precedence over LOCKMAP.

If RELEASE = T, then not only will the buffers containing the specified pages be unmapped, but the buffers themselves will be released, i.e. returned to the Interlisp storage manager.

(LOCKMAP PTR) [Function]  
For those situations in which a program needs prolonged access to a particular file page, LOCKMAP can be used to prevent MAPPAGE from shifting or unmapping the contents of the given core page. PTR is a pointer into a mapped page (i.e. a value of MAPWORD or MAPPAGE). LOCKMAP locks the indicated page in core until a corresponding UNLOCKMAP has been performed. If a page has been locked twice,

## Page Mapped Files

it must be unlocked twice before it is available for reuse. Returns `PTR` .

(`UNLOCKMAP PTR` )

[Function]

`PTR` is a pointer into a mapped page. `UNLOCKMAP` removes the most recent lock for that page.