

## 0.1 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.

Three functions are provided: `SETUPTIMER`, `SETUPTIMER.DATE`, and `TIMEREXPIRED?`. Also several new `is.oprs` have been defined: `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 first 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 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 specified 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` specifies 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` specifies 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 specified date and time. `DTS` is a Date/Time string such as `IDATE` accepts (page X.XX). If `OLDTIMER?` is a `Timer`, it will be reused and returned, rather than allocating a new `Timer`.

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

## Timers and Duration Functions

(TIMEREXPIRED? TIMER CLOCKV ALUE.OR.TIMER UNITS ) [Function]

If TIMER is a Timer, and CLOCKV ALUE.OR.TIMER UNITS is the time-unit of TIMER ,  
TIMEREXPIRED? returns true if TIMER has “gone o”.

CLOCKV ALUE.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 X.XX). 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.

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
  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 GLOBALRESOURCES 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 resourcename 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.

## Timers and Duration Functions