34. Dates and Times

The time package contains a set of functions for manipulating dates and times: finding the current time, reading and printing dates and times, converting between formats, and other miscellany regarding peculiarities of the calendar system. It also includes functions for accessing the Lisp Machine's microsecond timer.

Times are represented in two different formats by the functions in the time package. One way is to represent a time by many numbers, indicating a year, a month, a date, an hour, a minute, and a second (plus, sometimes, a day of the week and timezone). If a year less than 100 is specified, a multiple of 100 is added to it to bring it within 50 years of the present. Year numbers returned by the time functions are greater than 1900. The month is 1 for January, 2 for February, etc. The date is 1 for the first day of a month. The hour is a number from 0 to 23. The minute and second are numbers from 0 to 59. Days of the week are fixnums, where 0 means Monday, 1 means Tuesday, and so on. A timezone is specified as the number of hours west of GMT; thus in Massachusetts the timezone is 5. Any adjustment for daylight savings time is separate from this.

This "decoded" format is convenient for printing out times into a readable notation, but it is inconvenient for programs to make sense of these numbers and pass them around as arguments (since there are so many of them). So there is a second representation, called Universal Time, which measures a time as the number of seconds since January 1, 1900, at midnight GMT. This "encoded" format is easy to deal with inside programs, although it doesn't make much sense to look at (it looks like a huge integer). So both formats are provided; there are functions to convert between the two formats; and many functions exist in two versions, one for each format.

The Lisp Machine hardware includes a timer that counts once every microsecond. It is controlled by a crystal and so is fairly accurate. The absolute value of this timer doesn't mean anything useful, since it is initialized randomly; what you do with the timer is to read it at the beginning and end of an interval, and subtract the two values to get the length of the interval in microseconds. These relative times allow you to time intervals of up to an hour (32 bits) with microsecond accuracy.

The Lisp Machine keeps track of the time of day by maintaining a *timebase*, using the microsecond clock to count off the seconds. On the CADR, when the machine first comes up, the timebase is initialized by querying hosts on the Chaosnet to find out the current time. The Lambda has a calendar clock which never stops, so it normally does not need to do this. You can also set the time base using time:set-local-time, described below.

There is a similar timer that counts in 60ths of a second rather than microseconds; it is useful for measuring intervals of a few seconds or minutes with less accuracy. Periodic housekeeping functions of the system are scheduled based on this timer.

34.1 Getting and Setting the Time

get-decoded-time
time:get-time

Gets the current time, in decoded form. Return seconds, minutes, hours, date, month, year, day-of-the-week, and daylight-savings-time-p, with the same meanings as decode-universal-time (see page 782). If the current time is not known, nil is returned.

The name time:get-time is obsolete.

get-universal-time

Returns the current time in Universal Time form.

time:set-local-time &optional *new-time*

Sets the local time to *new-time*. If *new-time* is supplied, it must be either a universal time or a suitable argument to time:parse-universal-time (see page 781). If it is not supplied, or if there is an error parsing the argument, you are prompted for the new time. Note that you will not normally need to call this function; it is useful mainly when the timebase gets screwed up for one reason or another.

34.1.1 Elapsed Time in 60ths of a Second

The following functions deal with a different kind of time. These are not calendrical date/times, but simply elapsed time in 60ths of a second. These times are used for many internal purposes where the idea is to measure a small interval accurately, not to depend on the time of day or day of month.

time

Returns a number that increases by 1 every 60th of a second. The value wraps around roughly once a day. Use the time-lessp and time-difference functions to avoid getting in trouble due to the wrap-around. time is completely incompatible with the Maclisp function of the same name.

Note that time with an argument measures the length of time required to evaluate a form. See page 794.

get-internal-run-time
get-internal-real-time

Returns the total time in 60ths of a second since the last boot. This value does not wrap around. Eventually it becomes a bignum. The Lisp Machine does not distinguish between run time and real time.

internal-time-units-per-second *Constant*

According to Common Lisp, this is the ratio between a second and the time unit used by values of get-internal-real-time. On the Lisp Machine, the value is 60. The value may be different in other Common Lisp implementations.

time-lessp *time1 time2*

t if *time1* is earlier than *time2*, compensating for wrap-around, otherwise nil.

time-difference *time1 time2*

Assuming *time1* is later than *time2*, returns the number of 60ths of a second difference between them, compensating for wrap-around.

time-increment *time interval*

Increments *time* by *interval*, wrapping around if appropriate.

34.1.2 Elapsed Time in Microseconds

time:microsecond-time

Returns the value of the microsecond timer, as a bignum. The values returned by this function wrap around back to zero about once per hour.

time:fixnum-microsecond-time

Returns as a fixnum the value of the low 23 bits of the microsecond timer. This is like time:microsecond-time, with the advantage that it returns a value in the same format as the time function, except in microseconds rather than 60ths of a second. This means that you can compare fixnum-microsecond-times with time-lessp and time-difference. time:fixnum-microsecond-time is also a bit faster, but has the disadvantage that since you only see the low bits of the clock, the value can wrap around more quickly (about every eight seconds). Note that the Lisp Machine garbage collector is so designed that the bignums produced by time:microsecond-time are garbage-collected quickly and efficiently, so the overhead for creating the bignums is really not high.

34.2 Printing Dates and Times

The functions in this section create printed representations of times and dates in various formats and send the characters to a stream. To any of these functions, you may pass nil as the *stream* parameter and the function will return a string containing the printed representation of the time, instead of printing the characters to any stream.

The three functions time:print-time, time:print-universal-time, time:print-brief-universal-time and time:print-current-time accept an argument called *date-print-mode*, whose purpose is to control how the date is printed. It always defaults to the value of time:*default-date-print-mode*. Possible values include:

:dd//mm//yy	Print the date as in '3/16/53'.
:mm//dd//yy	Print as in '16/3/53'.
:dd-mm-yy	Print as in '16-3-53'.
:dd-mmm-yy	Print as in '16-Mar-53'.
:|dd mmm yy|	Print as in '16 Mar 53'.
:ddmmmyy	Print as in '16Mar53'.

:yymmdd	Print as in '530316'.
:yymmmdd	Print as in '53Mar16'.

time:print-current-time &optional (*stream* *standard-output*)
Prints the current time, formatted as in 11/25/80 14:50:02, to the specified stream. The date portion may be printed differently according to the argument *date-print-mode*.

time:print-time *seconds minutes hours date month year* &optional (*stream* *standard-output*) *date-print-mode*
Prints the specified time, formatted as in 11/25/80 14:50:02, to the specified stream. The date portion may be printed differently according to the argument *date-print-mode*.

time:print-universal-time *universal-time* &optional (*stream* *standard-output*) (*timezone* time:*timezone*) *date-print-mode*
Prints the specified time, formatted as in 11/25/80 14:50:02, to the specified stream. The date portion may be printed differently according to the argument *date-print-mode*.

time:print-brief-universal-time *universal-time* &optional (*stream* *standard-output*) *reference-time date-print-mode*
This is like time:print-universal-time except that it omits seconds and only prints those parts of *universal-time* that differ from *reference-time*, a universal time that defaults to the current time. Thus the output is in one of the following three forms:

```
02:59            ;the same day
3/4 14:01        ;a different day in the same year
8/17/74 15:30    ;a different year
```

The date portion may be printed differently according to the argument *date-print-mode*.

time:*default-date-print-mode* *Variable*
Holds the default for the *date-print-mode* argument to each of the functions above. Initially the value here is :mm//dd/yy.

time:print-current-date &optional (*stream* *standard-output*)
Prints the current time, formatted as in Tuesday the twenty-fifth of November, 1980; 3:50:41 pm, to the specified stream.

time:print-date *seconds minutes hours date month year day-of-the-week* &optional (*stream* *standard-output*)
Prints the specified time, formatted as in Tuesday the twenty-fifth of November, 1980; 3:50:41 pm, to the specified stream.

time:print-universal-date *universal-time* &optional (*stream* *standard-output*) (*timezone* time:*timezone*)
Prints the specified time, formatted as in Tuesday the twenty-fifth of November, 1980; 3:50:41 pm, to the specified stream.

34.3 Reading Dates and Times

These functions accept most reasonable printed representations of date and time and convert them to the standard internal forms. The following are representative formats that are accepted by the parser. Note that slashes are escaped with additional slashes, as is necessary if these strings are input in traditional syntax.

```
"March 15, 1960"        "3//15//60"    "3//15//1960"
"15 March 1960"         "15//3//60"    "15//3//1960"
"March-15-60"           "3-15-60"      "3-15-1960"
"15-March-60"           "15-3-60"      "15-3-1960"
"15-Mar-60"             "3-15"         "15 March 60"
"Fifteen March 60"     "The Fifteenth of March, 1960;"
"Friday, March 15, 1980"

"1130."      "11:30"      "11:30:17"   "11:30 pm"
"11:30 AM"   "1130"       "113000"
"11.30"      "11.30.00"   "11.3"       "11 pm"

"12 noon"    "midnight"   "m"      "6:00 gmt"   "3:00 pdt"
```

any date format may be used with any time format

```
"One minute after March 3, 1960"
```
meaning one minute after midnight
```
"Two days after March 3, 1960"
"Three minutes after 23:59:59 Dec 31, 1959"

"Now"      "Today"      "Yesterday"    "five days ago"
"two days after tomorrow"   "the day after tomorrow"
"one day before yesterday"  "BOB@OZ's birthday"
```

time:parse *string* &optional (*start* 0) (*end* nil) (*futurep* t) *base-time* *must-have-time* *date-must-have-year* *time-must-have-second* (*day-must-be-valid* t)

Interpret *string* as a date and/or time, and return seconds, minutes, hours, date, month, year, day-of-the-week, daylight-savings-time-p, and relative-p. *start* and *end* delimit a substring of the string; if *end* is nil, the end of the string is used. *must-have-time* means that *string* must not be empty. *date-must-have-year* means that a year must be explicitly specified. *time-must-have-second* means that the second must be specified. *day-must-be-valid* means that if a day of the week is given, then it must actually be the day that corresponds to the date. *base-time* provides the defaults for unspecified components; if it is nil, the current time is used. *futurep* means that the time should be interpreted as being in the future; for example, if the base time is 5:00 and the string refers to the time 3:00, that means the next day if *futurep* is non-nil, but it means two hours ago if *futurep* is nil. The *relative-p* returned value is t if the string included a relative part, such as 'one minute after' or 'two days before' or 'tomorrow' or 'now'; otherwise, it is nil.

If the input is not valid, the error condition **sys:parse-error** is signaled (see page 505).

time:parse-universal-time *string* &optional (*start* 0) (*end* nil) (*futurep* t) *base-time* *must-have-time* *date-must-have-year* *time-must-have-second* (*day-must-be-valid* t)

This is the same as **time:parse** except that it returns two values: an integer, representing the time in Universal Time, and the *relative-p* value.

34.4 Reading and Printing Time Intervals

In addition to the functions for reading and printing instants of time, there are other functions specifically for printing time intervals. A time interval is either a number (measured in seconds) or **nil**, meaning 'never'. The printed representations used look like '3 minutes 23 seconds' for actual intervals, or 'Never' for **nil** (some other synonyms and abbreviations for 'never' are accepted as input).

time:print-interval-or-never *interval* &optional (*stream* ***standard-output***)

interval should be a non-negative fixnum or **nil**. Its printed representation as a time interval is written onto *stream*.

time:parse-interval-or-never *string* &optional *start end*

Converts *string*, a printed representation for a time interval, into a number or *nil*. *start* and *end* may be used to specify a portion of *string* to be used; the default is to use all of *string*. It is an error if the contents of string do not look like a reasonable time interval. Here are some examples of acceptable strings:

```
"4 seconds"       "4 secs"          "4 s"
"5 mins 23 secs"  "5 m 23 s"        "23 SECONDS 5 M"
             "3 yrs 1 week 1 hr 2 mins 1 sec"
"never"           "not ever"        "no"              ""
```

Note that several abbreviations are understood, the components may be in any order, and case (upper versus lower) is ignored. Also, "months" are not recognized, since various months have different lengths and there is no way to know which month is being spoken of. This function always accepts anything that was produced by **time:print-interval-or-never**; furthermore, it returns exactly the same fixnum (or nil) that was printed.

time:read-interval-or-never &optional (*stream* ***standard-input***)

Reads a line of input from *stream* (using **readline**) and then calls **time:parse-interval-or-never** on the resulting string.

34.5 Time Conversions

decode-universal-time *universal-time* &optional (*timezone* time:*timezone*)

Converts *universal-time* into its decoded representation. The following values are returned: seconds, minutes, hours, date, month, year, day-of-the-week, daylight-savings-time-p, and the timezone used. *daylight-savings-time-p* tells you whether or not daylight savings time is in effect; if so, the value of *hour* has been adjusted accordingly. You can specify *timezone* explicitly if you want to know the equivalent representation for this time in other parts of the world.

encode-universal-time *seconds minutes hours date month year* &optional *timezone*

Converts the decoded time into Universal Time format, and return the Universal Time as an integer. If you don't specify *timezone*, it defaults to the current timezone adjusted for daylight savings time; if you provide it explicitly, it is not adjusted for daylight savings time. If *year* is less than 100, it is shifted by centuries until it is within 50 years of the present.

time:*timezone* *Variable*

The value of time:*timezone* is the time zone in which this Lisp Machine resides, expressed in terms of the number of hours west of GMT this time zone is. This value does not change to reflect daylight savings time; it tells you about standard time in your part of the world.

34.6 Internal Functions

These functions provide support for those listed above. Some user programs may need to call them directly, so they are documented here.

time:initialize-timebase

Initializes the timebase by querying Chaosnet hosts to find out the current time. This is called automatically during system initialization. You may want to call it yourself to correct the time if it appears to be inaccurate or downright wrong. See also **time:set-local-time**, page 777.

time:daylight-savings-time-p *hours date month year*

Returns t if daylight savings time is in effect for the specified hour; otherwise, return **nil**. If *year* is less than 100, it is shifted by centuries until it is within 50 years of the present.

time:daylight-savings-p

Returns t if daylight savings time is currently in effect; otherwise, returns **nil**.

time:month-length *month year*

Returns the number of days in the specified *month*; you must supply a *year* in case the month is February (which has a different length during leap years). If *year* is less than 100, it is shifted by centuries until it is within 50 years of the present.

time:leap-year-p *year*

Returns t if *year* is a leap year; otherwise return nil. If *year* is less than 100, it is shifted by centuries until it is within 50 years of the present.

time:verify-date *date month year day-of-the-week*

If the day of the week of the date specified by *date*, *month*, and *year* is the same as *day-of-the-week*, returns nil; otherwise, returns a string that contains a suitable error message. If *year* is less than 100, it is shifted by centuries until it is within 50 years of the present.

time:day-of-the-week-string *day-of-the-week* &optional (*mode* :long)

Returns a string representing the day of the week. As usual, 0 means Monday, 1 means Tuesday, and so on. Possible values of *mode* are:

:long	Returns the full English name, such as "Monday", "Tuesday", etc. This is the default.
:short	Returns a three-letter abbreviation, such as "Mon", "Tue", etc.
:medium	Returns a longer abbreviation, such as "Tues" and "Thurs".
:french	Returns the French name, such as "Lundi", "Mardi", etc.
:german	Returns the German name, such as "Montag", "Dienstag", etc.
:italian	Returns the Italian name, such as "Lunedi", "Martedi", etc.

time:month-string *month* &optional (*mode* :long)

Returns a string representing the month of the year. As usual, 1 means January, 2 means February, etc. Possible values of *mode* are:

:long	Returns the full English name, such as "January", "February", etc. This is the default.
:short	Returns a three-letter abbreviation, such as "Jan", "Feb", etc.
:medium	Returns a longer abbreviation, such as "Sept", "Novem", and "Decem".
:roman	Returns the Roman numeral for *month* (this convention is used in Europe).
:french	Returns the French name, such as "Janvier", "Fevrier", etc.
:german	Returns the German name, such as "Januar", "Februar", etc.
:italian	Returns the Italian name, such as "Gennaio", "Febbraio", etc.

time:timezone-string &optional (*timezone* time:*timezone*)
(*daylight-savings-p* (time:daylight-savings-p))

Return the three-letter abbreviation for this time zone. For example, if *timezone* is 5, then either "EST" (Eastern Standard Time) or "CDT" (Central Daylight Time) is used, depending on *daylight-savings-p*.