

The local-time Manual

Version 1.0.0

Daniel Lowe
Attila Lendvai

Copyright © 2012 Daniel Lowe <dlowe dlowe.net>
Copyright © 2012 Attila Lendvai <attila.lendvai gmail.com>

This manual describes the `local-time` Common Lisp library which is based on Erik Naggum's *The Long, Painful History of Time* [NaggumPaper] paper.

Table of Contents

1	Introduction.....	1
2	Public API.....	2
2.1	Types.....	2
2.2	Timezones	2
2.3	Creating <code>timestamp</code> Objects	2
2.4	Querying <code>timestamp</code> Objects.....	3
2.5	Manipulating Date and Time Values.....	5
2.6	Parsing and Formatting	6
3	Other Features.....	9
3.1	Reader Macros	9
3.2	Support for non-Gregorian Calendars.....	9
4	References	10
	Index.....	11

1 Introduction

The `local-time` library is a Common Lisp library for the manipulation of dates, times and intervals. It was originally based almost entirely upon Erik Naggum's paper *The Long Painful History of Time* [NaggumPaper]. Many of the core concepts originated from this paper, such as the separation of days and seconds, the choice of 2000-03-01 as the standard epoch, and the timestring format.

Caveats: This implementation assumes that time zone information is stored in the tzfile format. The default timezone is loaded from `/etc/localtime`. On non-POSIX systems, this will certainly give different results than the system time handling.

2 Public API

2.1 Types

It's a good idea to treat all values as immutable objects. `local-time` will not modify any object it was given unless explicitly asked to by the `:into` keyword argument.

timestamp *day sec nsec* [Class]

timestamp values can represent either a *date*, a *daytime* or a *time* value. It has the following slots:

```
(defclass timestamp ()
  ((day :type integer)
   (sec :type integer)
   (nsec :type (integer 0 999999999))))
```

The following constraints apply to the specific types:

- *date*: must have a `+utc-zone+` timezone and the *sec* slot must be the first second of a day; In other words, the time elements of the **timestamp** value must have their least possible values.
- *time*: the *day* slot must be zero.

timezone *path name loaded* [Struct]

timezone objects represent timezones - local and political modifications to the time representation. Timezones are responsible for storing offsets from GMT, abbreviations for different sub-timezones, and the times each sub-timezone is to be in effect.

2.2 Timezones

default-timezone [Default]

The variable `*default-timezone*` contains the timezone that will be used by default if none is specified. It is loaded from `/etc/localtime` when the library is loaded. If `/etc/localtime` is not present, it will default to UTC.

+utc-time+ [Constant]

The variable `+utc-zone+` contains a timezone corresponding to UTC.

define-timezone *zone-name zone-file &key* (*load nil*) [Macro]

Define *zone-name* (a symbol or a string) as a new timezone, lazy-loaded from *zone-file* (a pathname designator relative to the `zoneinfo` directory on this system. If *load* is true, load immediately.

2.3 Creating timestamp Objects

universal-to-timestamp *universal &key* (*nsec 0*) [Function]

Produces a **timestamp** instance from the provided universal time *universal*. Universal time is defined in the Common Lisp Specification as the number of seconds since 1900-01-01T00:00:00Z.

unix-to-timestamp *unix &key (nsec 0)* [Function]
 Produces a **timestamp** instance from the provided unix time *unix*. Unix time is defined by POSIX as the number of seconds since 1970-01-01T00:00:00Z.

now [Function]
 Produces a **timestamp** instance with the current time. Under sbcl, the new timestamp will be precise to the microsecond. Otherwise, the precision is limited to the second.

today [Function]
 Produces a **timestamp** instance that corresponds to today's date, which is the mid-night of the current day in the UTC zone.

encode-timestamp *nsec sec minute hour day month year &key timezone offset into* [Function]
 Returns a new **timestamp** instance corresponding to the specified time elements. The *offset* is the number of seconds offset from UTC of the locale. If *offset* is not specified, the offset will be guessed from the *timezone*. If a **timestamp** is passed as the *into* argument, its value will be set and that **timestamp** will be returned. Otherwise, a new **timestamp** is created.

make-timestamp *&key :day :sec :nsec* [Macro]
 Expands to an expression that creates an instance of a **timestamp** exactly as specified.

clone-timestamp *timestamp* [Macro]
 Expands to an expression that creates another copy of *timestamp* that is **timestamp**= to it.

2.4 Querying timestamp Objects

day-of *timestamp* [Function]
 Returns the day component of *timestamp*. Although Naggum's paper specifies that the day should be a signed fixnum, it is left unbounded for flexibility reasons.

sec-of *timestamp* [Function]
 Returns the 'seconds' component of the time. Valid values for the seconds range from 0 to 86399.

nsec-of *timestamp* [Function]
 Returns the 'microseconds' component of the time. Valid values for the nanoseconds range from 0 to 999999999.

timestamp-to-universal *timestamp* [Function]
 This returns the date/time specified in *timestamp* encoded as the number of seconds since January 1st, 1900 12:00am UTC.

timestamp-to-unix *timestamp* [Function]
 This returns the date/time specified in *timestamp* encoded as the number of seconds since January 1st, 1970 12:00am UTC. It corresponds with the time received from the POSIX call `time()`.

timestamp-subtimezone *timestamp timezone* [Function]

Returns as multiple values the time zone applicable at the given time as the number of seconds east of UTC, a boolean daylight-saving-p, and the customary abbreviation of the timezone.

with-decoded-timestamp (**&key** *nsec sec minute hour day month year day-of-week daylight-p timezone*) *timestamp &body body* [Macro]

This macro binds variables to the decoded elements of *timestamp*. The *timezone* argument is used for decoding the timestamp, and is not bound by the macro. The value of *day-of-week* starts from 0 which means Sunday.

decode-timestamp *timestamp* [Function]

Returns the decoded time as (values *ns ss mm hh day month year day-of-week daylight-saving-time-p timezone-offset timezone-abbreviation*).

timestamp< *time-a time-b* [Function]

timestamp<= *time-a time-b* [Function]

timestamp> *time-a time-b* [Function]

timestamp>= *time-a time-b* [Function]

timestamp= *time-a time-b* [Function]

timestamp/= *time-a time-b* [Function]

These comparison functions act like their string and char counterparts.

timestamp-minimum *timestamp &rest timestamps* [Function]

Returns the earliest timestamp passed to it.

timestamp-maximum *timestamp &rest timestamps* [Function]

Returns the latest timestamp passed to it.

timestamp-day-of-week *timestamp* [Function]

This returns the index of the day of the week, starting at 0 which means Sunday.

Note: "Day of the week" is ambiguous and locale dependent.

universal-to-timestamp *timestamp* [Function]

Returns the UNIVERSAL-TIME corresponding to *timestamp*.

Note: Subsecond precision is not preserved.

timestamp-millennium *timestamp &key timezone* [Function]

timestamp-century *timestamp &key timezone* [Function]

timestamp-decade *timestamp &key timezone* [Function]

Returns the ordinal millennium, century or decade upon which the timestamp falls.

Ordinal time values start at 1, so the (timestamp-century (now)) will return 21.

timestamp-year *timestamp &key timezone* [Function]

timestamp-month *timestamp &key timezone* [Function]

timestamp-day *timestamp &key timezone* [Function]

timestamp-hour *timestamp &key timezone* [Function]

timestamp-minute *timestamp &key timezone* [Function]

timestamp-second *timestamp &key timezone* [Function]

`timestamp-millisecond` *timestamp &key timezone* [Function]
`timestamp-microsecond` *timestamp &key timezone* [Function]
`timestamp-microsecond` *timestamp &key timezone* [Function]
 Returns the decoded part of the timestamp.

2.5 Manipulating Date and Time Values

`timestamp+` *time amount unit* [Function]
`timestamp-` *time amount unit* [Function]

Add or subtract the *amount* to the *time* using the specified *unit*. *unit* may be one of (`:nsec` `:sec` `:minute` `:hour` `:day` `:month` `:year`). The value of the parts of the timestamp of higher resolution than the UNIT will never be touched. If you want a precise number of seconds from a time, you should specify the offset in seconds.

`timestamp-maximize-part` *timestamp part &key offset timezone into* [Function]
 Returns a timestamp with its parts maximized up to *part*. *part* can be any of (`:nsec` `:sec` `:min` `:hour` `:day` `:month`). If *into* is specified, it will be modified and returned, otherwise a new timestamp will be created.

`timestamp-minimize-part` *timestamp part &key offset timezone into* [Function]
 Returns a timestamp with its parts minimized up to *part*. *part* can be any of (`:nsec` `:sec` `:min` `:hour` `:day` `:month`). If *into* is specified, it will be modified and returned, otherwise a new timestamp will be created.

`adjust-timestamp` *timestamp &body changes* [Macro]
 Alters various parts of *timestamp*, given a list of changes. The changes are in the format (offset part value) and (set part value).

;; Return a new timestamp value that points to the previous Monday
 (adjust-timestamp (today) (offset :day-of-week :monday))

;; Return a new timestamp value that points three days ahead from now
 (adjust-timestamp (today) (offset :day 3))

Keep in mind that `adjust-timestamp` is not a mere setter for fields but instead it handles overflows and timezone conversions as expected. Also note that it's possible to specify multiple commands.

The list of possible places to manipulate are: `:nsec` `:sec` `:sec-of-day` `:minute` `:hour` `:day` `:day-of-month` `:month` `:year`.

`adjust-timestamp!` *timestamp &body changes* [Macro]
 Just like `adjust-timestamp`, but instead of returning a freshly constructed value, it alters the provided *timestamp* value (and returns it).

`timestamp-whole-year-difference` *time-a time-b* [Function]
 Returns the number of whole years elapsed between *time-a* and *time-b*.

Note: This is useful for calculating anniversaries and birthdays.

`days-in-month` *month year* [Function]
 Returns the number of days in a given month of the specified year.

2.6 Parsing and Formatting

+iso-8601-format+ [Constant]

The constant **+iso-8601-format+** is bound to a description of the ISO 8601 format. An output with this format will look like this: `'2008-03-01T19:42:34.608506+01:00'`. This is the default format for the **format-timestring** function.

+asctime-format+ [Constant]

The constant **+asctime-format+** is bound to a format mirroring the output of the POSIX `asctime()` function. An output with this format will look like this: `'Sat Mar 1 19:42:34 2008'`.

+rfc-1123-format+ [Constant]

The constant **+rfc-1123-format+** is bound to a description of the format defined in RFC 1123 for Internet timestamps. An output with this format will look like this: `'Sat, 01 Mar 2008 19:42:34 EDT'`.

+iso-week-date-format+ [Constant]

The constant **+iso-week-date-format+** is bound to a description of the ISO 8601 Week Date format. An output with this format will look like this: `'2009-W53-5'`.

parse-timestring *timestring* **&key** (*start* **0**) *end* (*fail-on-error* **t**) (*offset* **0**) [Function]

Parses a timestring and returns the corresponding **timestamp**. Parsing begins at *start* and stops at the *end* position. If there are invalid characters within *timestring* and *fail-on-error* is **T**, then an **invalid-timestring** error is signaled, otherwise **NIL** is returned.

If there is no timezone specified in *timestring* then *offset* is used as the default timezone offset (in seconds).

format-timestring (*destination* *timestamp* **&key** (*format* **+iso-8601-format+**) (*timezone* **default-timezone**)) [Function]

Constructs a string representation of **TIMESTAMP** according to **FORMAT** and returns it. If *destination* is **T**, the string is written to **standard-output**. If *destination* is a stream, the string is written to the stream.

FORMAT is a list containing one or more of strings, characters, and keywords. Strings and characters are output literally, while keywords are replaced by the values here:

:year	<i>*year</i>
:month	<i>*numeric month</i>
:day	<i>*day of month</i>
:weekday	<i>*numeric day of week, starting from 0 which means Sunday</i>
:hour	<i>*hour</i>
:min	<i>*minutes</i>
:sec	<i>*seconds</i>
:msec	<i>*milliseconds</i>

```

:usec      *microseconds
:nsec      *nanoseconds

:iso-week-year
      *year for ISO week date (can be different from regular calendar year)

:iso-week-number
      *ISO week number (i.e. 1 through 53)

:iso-week-day
      *ISO compatible weekday number (i.e. monday=1, sunday=7)

:ordinal-day
      day of month as an ordinal (e.g. 1st, 23rd)

:long-weekday
      long form of weekday (e.g. Sunday, Monday)

:short-weekday
      short form of weekday (e.g. Sun, Mon)

:long-month
      long form of month (e.g. January, February)

:short-month
      short form of month (e.g. Jan, Feb)

:hour12    hour on a 12-hour clock

:ampm      am/pm marker in lowercase

:gmt-offset
      the gmt-offset of the time, in +00:00 form

:gmt-offset-or-z
      like :gmt-offset, but is Z when UTC

:timezone
      timezone abbreviation for the time

```

Elements marked by * can be placed in a list in the form:

```
(:keyword padding &optional (padchar #\0))
```

The string representation of the value will be padded with the padchar.

You can see examples by examining the values in `+iso-8601-format+`, `+asctime-format+`, and `+rfc-1123-format+`.

Produces on *stream* the timestring corresponding to the *timestamp* with the given options. If *stream* is `nil`, only returns a string containing what would have been the output. If *stream* is `t`, prints the string to **standard-output**.

Example output:

```

LOCAL-TIME> (format-timestring nil (now))
"2008-03-01T19:42:34.608506+01:00"

```

format-rfc3339-timestring (*destination timestamp &key* [Function]
omit-date-part omit-time-part omit-timezone-part (use-zulu t))
Formats the time like `format-timestring`, but in RFC 3339 format. The options control valid options in the RFC.

3 Other Features

3.1 Reader Macros

`enable-read-macros`

[Function]

Adds `@TIMESTRING` and `#@UNIVERSAL-TIME` as reader macros.

3.2 Support for non-Gregorian Calendars

`astronomical-julian-date` *timestamp*

[Function]

Returns the julian date of the date portion of *timestamp*.

`astronomical-julian-date` *timestamp*

[Function]

Returns the modified julian date of the date portion of *timestamp*.

4 References

- [NaggumPaper] Erik Naggum. *The Long Painful History of Time* <http://naggum.no/lugm-time.html>, 1999.

Index

*

default-timezone 2

+

+asctime-format+ 6
 +iso-8601-format+ 6
 +iso-week-date-format+ 6
 +rfc-1123-format+ 6
 +utc-time+ 2

A

adjust-timestamp 5
 adjust-timestamp! 5
 astronomical-julian-date 9

C

clone-timestamp 3

D

day-of 3
 days-in-month 5
 decode-timestamp 4
 define-timezone 2

E

enable-read-macros 9
 encode-timestamp 3

F

format-rfc3339-timestring 8
 format-timestring 6

M

make-timestamp 3
 modified-julian-date 9

N

now 3
 nsec-of 3

P

parse-timestring 6

S

sec-of 3

T

timestamp 2
 timestamp+ 5
 timestamp- 5
 timestamp-century 4
 timestamp-day 4
 timestamp-day-of-week 4
 timestamp-decade 4
 timestamp-hour 4
 timestamp-maximize-part 5
 timestamp-maximum 4
 timestamp-microsecond 4
 timestamp-millennium 4
 timestamp-millisecond 4
 timestamp-minimize-part 5
 timestamp-minimum 4
 timestamp-minute 4
 timestamp-month 4
 timestamp-second 4
 timestamp-subtimezone 4
 timestamp-to-universal 3
 timestamp-to-unix 3
 timestamp-whole-year-difference 5
 timestamp-year 4
 timestamp/= 4
 timestamp< 4
 timestamp<= 4
 timestamp= 4
 timestamp> 4
 timestamp>= 4
 today 3

U

universal-to-timestamp 2, 4
 unix-to-timestamp 3

W

with-decoded-timestamp 4