File created: 18-Oct-93 15:20:26 **{Pele:mv:envos}<LispCore>Sources>CLTL2>CMLTIME.;2**

previous date: 3-Sep-91 17:50:39 {Pele:mv:envos}<LispCore>Sources>CLTL2>CMLTIME.;1

Read Table: XCL

Package: INTERLISP

Format: XCCS


```
(RPAQQ CMLTIMECOMS
       (
```

;;; Common Lisp Time Functions

```
        (FUNCTIONS %CONVERT-INTERNAL-TIME-TO-CLUT)
        (CONSTANTS (CL:INTERNAL-TIME-UNITS-PER-SECOND 1000))
        (FNS CL:GET-INTERNAL-REAL-TIME CL:GET-INTERNAL-RUN-TIME CL:GET-UNIVERSAL-TIME CL:GET-DECODED-TIME
             CL:DECODE-UNIVERSAL-TIME CL:ENCODE-UNIVERSAL-TIME CL:SLEEP)
        (PROP FILETYPE CMLTIME)
        (DECLARE\: DONTEVAL@LOAD DOEVAL@COMPILE DONTCOPY COMPILERVARS
               (ADDVARS (NLAMA)
                        (NLAML)
                        (LAMA CL:SLEEP CL:ENCODE-UNIVERSAL-TIME CL:DECODE-UNIVERSAL-TIME CL:GET-DECODED-TIME
                              CL:GET-UNIVERSAL-TIME CL:GET-INTERNAL-RUN-TIME)))))
```

;;; Common Lisp Time Functions

```
(DEFMACRO %CONVERT-INTERNAL-TIME-TO-CLUT (TIME)
    ;; Converts from Interlisp-D internal time format to Common Lisp Universal Time
    `(+ ,TIME (CL:* 365 24 60 60)
        MAX.FIXP 1))

(DECLARE\: EVAL@COMPILE

(RPAQQ CL:INTERNAL-TIME-UNITS-PER-SECOND 1000)

(CONSTANTS (CL:INTERNAL-TIME-UNITS-PER-SECOND 1000))
)

(DEFINEQ
```

(CL:GET-INTERNAL-REAL-TIME

```
  (LAMBDA NIL                                                    (* |hdj| "18-Jul-86 12:05")
```

;;; The current time is returned as a single integer in Internal Time format. (Internal Time format = time in milliseconds for us.) This time is relative to an
;;; arbitrary time base, but the difference between the values of two calls to this function will be the amount of elapsed real time between the two calls,
;;; measured in the units defined by INTERNAL-TIME-UNITS-PER-SECOND

```
    (CLOCK 0)))
```

(CL:GET-INTERNAL-RUN-TIME

```
  (CL:LAMBDA NIL                                                 (* |hdj| "18-Jul-86 12:06")
```

;;; The current run time is returned as a single integer in Internal Time format. (Internal Time format = time in milliseconds for us.) The precise meaning
;;; of this quantity is implementation-dependent; it may measure real time, run time, CPU cycles, or some other quantity. The intent is that the difference
;;; between the values of two calls to this function be the amount of time between the two calls during which the computational effort was expended on
;;; behalf of the executing program.

```
          (CLOCK 2)))
```

(CL:GET-UNIVERSAL-TIME

```
  (CL:LAMBDA NIL                                                 (* |hdj| "18-Jul-86 12:02")
```

;;; The current time of day is returned as a single integer in Universal Time format.

```
          (%CONVERT-INTERNAL-TIME-TO-CLUT (DAYTIME))))
```

(CL:GET-DECODED-TIME

```
  (CL:LAMBDA NIL                                                 (* |hdj| "18-Jul-86 12:08")
```

;;; The current time is returned in Decoded Time format. Nine values are returned: SECOND, MINUTE, HOUR, DATE, MONTH, YEAR, DAY-OF-WEEK,
;;; DAYLIGHT-SAVING-TIME-P, and TIME-ZONE.

```
          (CL:DECODE-UNIVERSAL-TIME (CL:GET-UNIVERSAL-TIME))))
```

(CL:DECODE-UNIVERSAL-TIME

```
  (CL:LAMBDA (UNIVERSAL-TIME &OPTIONAL (TIME-ZONE |\\TimeZoneComp| TIME-ZONE-SUPPLIEDP))
                                                                 (* |kbr:| " 7-Aug-86 10:21")
```

;;; The time specified by UNIVERSAL-TIME in Universal Time format is converted to Decoded Time format. Nine values are returned: SECOND, ;;; MINUTE, HOUR, DATE, MONTH, YEAR, DAY-OF-WEEK, DAYLIGHT-SAVING-TIME-P, and TIME-ZONE.

```
(PROG (CHECKDLS TIME MONTH SEC HR TOTALDAYS DAYS LEAP400 LEAP100 LEAP4 YEAR YDAY WDAY MIN DLS)
```

;; Page 446 of the silver book: If you don't specify TIME-ZONE it defaults to the current time zone adjusted for daylight savings time. If you ;; provide TIME-ZONE explicitly, no adjustment for daylight savings time is is performed.

```
      (SETQ CHECKDLS (AND (NOT TIME-ZONE-SUPPLIEDP)
                          |\\DayLightSavings|))
      (CL:MULTIPLE-VALUE-SETQ (TIME SEC)
             (CL:FLOOR UNIVERSAL-TIME 60))
      (CL:MULTIPLE-VALUE-SETQ (TIME MIN)
             (CL:FLOOR TIME 60))
      (CL:MULTIPLE-VALUE-SETQ (TOTALDAYS HR)
             (CL:FLOOR (- TIME TIME-ZONE)
                    24))
 DTLOOP
```

;; LEAP400 = number of 400 year blocks till Jan 1, 2000 Note: The algorithm still works correctly for dates after Jan 1, 2000 . LEAP400 will ;; be negative but not wrong. (Any Jan 1 a year a multiple of 400 would do nicely. Jan 1, 2000 just happens to be close by.)

```
      (CL:MULTIPLE-VALUE-SETQ (LEAP400 DAYS)
             (CL:FLOOR (- 36524 TOTALDAYS)
                    (+ 36525 (CL:* 3 36524))))          ; LEAP100 = number of 100 year blocks till the 400 year blocks.
      (CL:MULTIPLE-VALUE-SETQ (LEAP100 DAYS)
             (CL:FLOOR DAYS 36524))                     ; LEAP4 = number of 4 year blocks till the 100 year blocks.
      (CL:MULTIPLE-VALUE-SETQ (LEAP4 DAYS)
             (CL:FLOOR DAYS (+ 366 (CL:* 3 365))))
```

;; Date of answer will be (+ (* 146097 LEAP400) (* 36524 LEAP100) (* 1461 LEAP4) DAYS) days before Jan 1, 2000

```
      (SETQ YEAR (- 2000 (CL:* 400 LEAP400)
                    (CL:* 100 LEAP100)
                    (CL:* 4 LEAP4)
                    (CDR (\\DTSCAN DAYS '((1096 . 4)
                                          (731 . 3)
                                          (366 . 2)
                                          (1 . 1)
                                          (0 . 0))))))
```

;; YDAY is the ordinal of day as it would appear in a leap year. We thus have Jan 1 = day 0, Feb 29 = day 59, Mar 1 = day 60, and Dec 31 ;; = day 365.

```
      (SETQ YDAY (- (CDR (\\DTSCAN DAYS (COND
                                          ((AND (EQ (CL:MOD YEAR 100)
                                                    0)
                                                (NOT (EQ (CL:MOD YEAR 400)
                                                         0)))
                                           '((1402 . 1460)
                                             (1096 . 1461)
                                             (1037 . 1095)
                                             (731 . 1096)
                                             (672 . 730)
                                             (366 . 731)
                                             (307 . 365)
                                             (1 . 366)
                                             (0 . 0)))
                                          (T '((1096 . 1461)
                                               (1037 . 1095)
                                               (731 . 1096)
                                               (672 . 730)
                                               (366 . 731)
                                               (307 . 365)
                                               (1 . 366)
                                               (0 . 0))))))
                    DAYS))
      (SETQ WDAY (CL:MOD TOTALDAYS 7))
      (COND
         ((AND CHECKDLS (SETQ DLS (\\ISDST? YDAY HR WDAY)))
```

;; This date is during daylight savings, so add 1 hour. Third arg is day of the week, which we determine by taking days mod 7 ;; plus offset. Monday = zero in this scheme. Jan 1 1900 was a Monday=0 so offset is 0

```
          (COND
             ((> (SETQ HR (CL:1+ HR))
                 23)
```

;; overflowed into the next day. This case is too hard (we might have overflowed the month, for example), so just go back ;; and recompute

```
              (SETQ TOTALDAYS (CL:1+ TOTALDAYS))
              (SETQ HR 0)
              (SETQ CHECKDLS NIL)
              (GO DTLOOP)))))
      (SETQ MONTH (\\DTSCAN YDAY '((335 . 12)
                                   (305 . 11)
                                   (274 . 10)
                                   (244 . 9)
                                   (213 . 8)
                                   (182 . 7)
```

```
                                    (152 . 6)
                                    (121 . 5)
                                    (91 . 4)
                                    (60 . 3)
                                    (31 . 2)
                                    (0 . 1))))          ; Now return (SECOND MINUTE HOUR DAY MONTH YEAR
                                                        ; WEEKDAY DAYLIGHT ZONE)
        (RETURN (CL:VALUES SEC MIN HR (CL:1+ (- YDAY (CAR MONTH)))
                           (CDR MONTH)
                           YEAR WDAY DLS TIME-ZONE)))))
```

(CL:ENCODE-UNIVERSAL-TIME

```
  (CL:LAMBDA (SECOND MINUTE HOUR DATE MONTH YEAR &OPTIONAL TIME-ZONE)
                                                        ; Edited 27-Oct-87 19:11 by bvm:
```

;;; The time specified by the given components of Decoded Time format is encoded into Universal Time format and returned. If you don't specify
;;; TIME-ZONE, it defaults to the current time zone adjusted for daylight saving time. If you provide TIME-ZONE explicitly, no adjustment for daylight
;;; saving time is performed.

```
    (PROG (YDAY DAYSSINCEDAY0)
```

;; From pages 444 and 445 of the silver book and Lucid testing, here are three examples of ENCODE-UNIVERSAL-TIME usage known to
;; be correct and which should be rechecked by anyone who edits this function: (ENCODE-UNIVERSAL-TIME 1 0 0 1 1 1900 0) = 1
;; (ENCODE-UNIVERSAL-TIME 1 0 0 1 1 1976 0) = 2398291201 (ENCODE-UNIVERSAL-TIME 0 0 0 1 1 3000 0) = 34712668800

;; If the YEAR is between 0 and 99 we have to figure out what the 'obvious' year is.

```
      (SETQ YEAR (CL:IF (< YEAR 100)
                        (CL:MULTIPLE-VALUE-BIND (SEC MIN HOUR DAY MONTH NOW-YEAR)
                            (CL:GET-DECODED-TIME)
                          (DECLARE (IGNORE SEC MIN HOUR DAY MONTH))
                          (CL:DO ((Y (+ YEAR (CL:* 100 (CL:1- (CL:TRUNCATE NOW-YEAR 100))))
                                     (+ Y 100)))
                                 ((<= (ABS (- Y NOW-YEAR))
                                      50)
                                  Y)))
                        YEAR))
      (SETQ YDAY (+ (SELECTQ MONTH
                          (1 0)
                          (2 31)
                          (3 59)
                          (4 90)
                          (5 120)
                          (6 151)
                          (7 181)
                          (8 212)
                          (9 243)
                          (10 273)
                          (11 304)
                          (12 334)
                          NIL)
                    (SUB1 DATE)))
      (SETQ DAYSSINCEDAY0 (+ YDAY (TIMES 365 (SETQ YEAR (IDIFFERENCE YEAR 1900)))
                             (IQUOTIENT (SUB1 YEAR)
                                        4)))
      (|if| (> MONTH 2)
          |then|                                        ; After February 28
                (|add| YDAY 1)                          ; Day-of-year is based on 366-day year
                (|if| (AND (EQ 0 (IREMAINDER YEAR 4))
                           (OR (NOT (EQ (IREMAINDER YEAR 100)
                                        0))
                               (EQ (IREMAINDER YEAR 400)
                                   0)))
                     |then|                             ; It is a leap year, so real day count also incremented
                           (|add| DAYSSINCEDAY0 1)))
```

;; This is almost right - now correct for 100/400 leap year rule. 1900 is magically handled by above formula, and 2000 is a leap year, so we
;; only need to do this for years after 2100.

```
      (FOR I FROM 200 TO YEAR BY 100 UNLESS (OR (= I YEAR)
                                                (EQ (IREMAINDER I 400)
                                                    100))
         DO (CL:DECF DAYSSINCEDAY0))
      (SETQ HOUR (+ HOUR (TIMES 24 DAYSSINCEDAY0)
                    (COND
                       (TIME-ZONE TIME-ZONE)
                       ((AND |\\DayLightSavings| (\\ISDST? YDAY HOUR (IREMAINDER DAYSSINCEDAY0 7)))
                        ;; Subtract one to go from daylight to standard time.  Weekday argument (IREMAINDER DAYSSINCEDAY0
                        ;; 7) to \ISDST?  is based on day 0 = Jan 1, 1900, which was a Monday = 0
                        (SUB1 |\\TimeZoneComp|))
                       (T |\\TimeZoneComp|))))
      (RETURN (+ SECOND (TIMES 60 (+ MINUTE (TIMES 60 HOUR)))))))
```

(CL:SLEEP

```
  (CL:LAMBDA (CL::SECONDS)                              ; Edited 24-Apr-87 15:28 by jrb:
```

;;; (SLEEP N) causes execution to cease and become dormant for approximately N seconds of real time, whereupon execution is resumed. The

;;; argument may be any non-negative non-complex number. SLEEP returns NIL.

```
         (DISMISS (ROUND (CL:* CL::SECONDS 1000)))
         NIL))

)

(PUTPROPS CMLTIME FILETYPE CL:COMPILE-FILE)

(DECLARE\: DONTEVAL@LOAD DOEVAL@COMPILE DONTCOPY COMPILERVARS

(ADDTOVAR NLAMA )

(ADDTOVAR NLAML )

(ADDTOVAR LAMA CL:SLEEP CL:ENCODE-UNIVERSAL-TIME CL:DECODE-UNIVERSAL-TIME CL:GET-DECODED-TIME
                    CL:GET-UNIVERSAL-TIME CL:GET-INTERNAL-RUN-TIME)
)

(PUTPROPS CMLTIME COPYRIGHT ("Venue & Xerox Corporation" 1986 1987 1990 1991 1993))
```

FUNCTION INDEX

PROPERTY INDEX

CONSTANT INDEX

MACRO INDEX