JFP **13** (1): 225–230, January 2003. © 2003 Cambridge University Press
DOI: 10.1017/S0956796803002612 Printed in the United Kingdom

Chapter 24

Dates and Times

```
module Time (
        ClockTime,
        Month(January,February,March,April,May,June,
              July,August,September,October,November,December),
        Day(Sunday,Monday,Tuesday,Wednesday,Thursday,Friday,Saturday),
        CalendarTime(CalendarTime, ctYear, ctMonth, ctDay, ctHour, ctMin,
                     ctSec, ctPicosec, ctWDay, ctYDay,
                     ctTZName, ctTZ, ctIsDST),
        TimeDiff(TimeDiff, tdYear, tdMonth, tdDay, tdHour,
                 tdMin, tdSec, tdPicosec),
        getClockTime, addToClockTime, diffClockTimes,
        toCalendarTime, toUTCTime, toClockTime,
        calendarTimeToString, formatCalendarTime ) where

import Ix(Ix)

data ClockTime = ...                    -- Implementation-dependent
instance Ord  ClockTime where ...
instance Eq   ClockTime where ...

data Month =  January   | February | March    | April
           |  May       | June     | July     | August
           |  September | October  | November | December
           deriving (Eq, Ord, Enum, Bounded, Ix, Read, Show)

data Day   =  Sunday | Monday  | Tuesday  | Wednesday | Thursday
           |  Friday | Saturday
           deriving (Eq, Ord, Enum, Bounded, Ix, Read, Show)
```

```
data CalendarTime = CalendarTime {
                ctYear                          :: Int,
                ctMonth                         :: Month,
                ctDay, ctHour, ctMin, ctSec     :: Int,
                ctPicosec                       :: Integer,
                ctWDay                          :: Day,
                ctYDay                          :: Int,
                ctTZName                        :: String,
                ctTZ                            :: Int,
                ctIsDST                         :: Bool
        } deriving (Eq, Ord, Read, Show)

data TimeDiff = TimeDiff {
                tdYear, tdMonth, tdDay, tdHour, tdMin, tdSec :: Int,
                tdPicosec                                    :: Integer
        } deriving (Eq, Ord, Read, Show) -- Functions on times
getClockTime         :: IO ClockTime

addToClockTime       :: TimeDiff  -> ClockTime -> ClockTime
diffClockTimes       :: ClockTime -> ClockTime -> TimeDiff

toCalendarTime       :: ClockTime    -> IO CalendarTime
toUTCTime            :: ClockTime    -> CalendarTime
toClockTime          :: CalendarTime -> ClockTime
calendarTimeToString :: CalendarTime -> String
formatCalendarTime   :: TimeLocale -> String -> CalendarTime -> String
```

The `Time` library provides standard functionality for clock times, including timezone information. It follows RFC 1129 in its use of Coordinated Universal Time (UTC).

`ClockTime` is an abstract type, used for the system's internal clock time. Clock times may be compared directly or converted to a calendar time `CalendarTime` for I/O or other manipulations. `CalendarTime` is a user-readable and manipulable representation of the internal `ClockTime` type. The numeric fields have the following ranges.

Value	Range	Comments
ctYear	-maxInt . . . maxInt	Pre-Gregorian dates are inaccurate
ctDay	1 . . . 31	
ctHour	0 . . . 23	
ctMin	0 . . . 59	
ctSec	0 . . . 61	Allows for two Leap Seconds
ctPicosec	0 . . . $(10^{12}) - 1$	
ctYDay	0 . . . 365	364 in non-Leap years
ctTZ	-89999 . . . 89999	Variation from UTC in seconds

The $ctTZName$ field is the name of the time zone. The $ctIsDST$ field is `True` if Daylight Savings Time would be in effect, and `False` otherwise. The `TimeDiff` type records the difference

https://doi.org/10.1017/S0956796803002612 Published online by Cambridge University Press

between two clock times in a user-readable way.

Function `getClockTime` returns the current time in its internal representation. The expression `addToClockTime` d t adds a time difference d and a clock time t to yield a new clock time. The difference d may be either positive or negative. The expression `diffClockTimes` $t1$ $t2$ returns the difference between two clock times $t1$ and $t2$ as a `TimeDiff`.

Function `toCalendarTime` t converts t to a local time, modified by the timezone and daylight savings time settings in force at the time of conversion. Because of this dependence on the local environment, `toCalendarTime` is in the `IO` monad.

Function `toUTCTime` t converts t into a `CalendarTime` in standard UTC format. `toClockTime` l converts l into the corresponding internal `ClockTime` ignoring the contents of the $ctWDay$, $ctYDay$, $ctTZName$, and $ctIsDST$ fields.

Function `calendarTimeToString` formats calendar times using local conventions and a formatting string.

24.1 Library `Time`

```
module Time (
        ClockTime,
        Month(January,February,March,April,May,June,
              July,August,September,October,November,December),
        Day(Sunday,Monday,Tuesday,Wednesday,Thursday,Friday,Saturday),
        CalendarTime(CalendarTime, ctYear, ctMonth, ctDay, ctHour, ctMin,
                     ctSec, ctPicosec, ctWDay, ctYDay,
                     ctTZName, ctTZ, ctIsDST),
        TimeDiff(TimeDiff, tdYear, tdMonth, tdDay,
                 tdHour, tdMin, tdSec, tdPicosec),
        getClockTime, addToClockTime, diffClockTimes,
        toCalendarTime, toUTCTime, toClockTime,
        calendarTimeToString, formatCalendarTime ) where

import Ix(Ix)
import Locale(TimeLocale(..),defaultTimeLocale)
import Char ( intToDigit )

data ClockTime = ...                    -- Implementation-dependent
instance Ord  ClockTime where ...
instance Eq   ClockTime where ...

data Month =  January   | February | March    | April
           |  May       | June     | July     | August
           |  September | October  | November | December
           deriving (Eq, Ord, Enum, Bounded, Ix, Read, Show)

data Day   =  Sunday | Monday  | Tuesday  | Wednesday | Thursday
           |  Friday | Saturday
           deriving (Eq, Ord, Enum, Bounded, Ix, Read, Show)
```

https://doi.org/10.1017/S0956796803002612 Published online by Cambridge University Press

```
data CalendarTime = CalendarTime {
                ctYear                          :: Int,
                ctMonth                         :: Month,
                ctDay, ctHour, ctMin, ctSec     :: Int,
                ctPicosec                       :: Integer,
                ctWDay                          :: Day,
                ctYDay                          :: Int,
                ctTZName                        :: String,
                ctTZ                            :: Int,
                ctIsDST                         :: Bool
        } deriving (Eq, Ord, Read, Show)

data TimeDiff = TimeDiff {
                tdYear, tdMonth, tdDay, tdHour, tdMin, tdSec :: Int,
                tdPicosec                                    :: Integer
        } deriving (Eq, Ord, Read, Show)

getClockTime            :: IO ClockTime
getClockTime            = ...           -- Implementation-dependent

addToClockTime          :: TimeDiff     -> ClockTime -> ClockTime
addToClockTime td ct    =  ...          -- Implementation-dependent

diffClockTimes          :: ClockTime    -> ClockTime -> TimeDiff
diffClockTimes ct1 ct2  =  ...          -- Implementation-dependent

toCalendarTime          :: ClockTime    -> IO CalendarTime
toCalendarTime ct       =  ...          -- Implementation-dependent

toUTCTime               :: ClockTime    -> CalendarTime
toUTCTime ct            =  ...          -- Implementation-dependent

toClockTime             :: CalendarTime -> ClockTime
toClockTime cal         =  ...          -- Implementation-dependent

calendarTimeToString    :: CalendarTime -> String
calendarTimeToString    =  formatCalendarTime defaultTimeLocale "%c"
```

https://doi.org/10.1017/S0956796803002612 Published online by Cambridge University Press

```
formatCalendarTime :: TimeLocale -> String -> CalendarTime -> String
formatCalendarTime l fmt ct@(CalendarTime year mon day hour min sec sdec
                                          wday yday tzname _ _) =
       doFmt fmt
  where doFmt ('%':c:cs) = decode c ++ doFmt cs
       doFmt (c:cs) = c : doFmt cs
       doFmt "" = ""

       to12 :: Int -> Int
       to12 h = let h' = h `mod` 12 in if h' == 0 then 12 else h'

       decode 'A' = fst (wDays l  !! fromEnum wday)
       decode 'a' = snd (wDays l  !! fromEnum wday)
       decode 'B' = fst (months l !! fromEnum mon)
       decode 'b' = snd (months l !! fromEnum mon)
       decode 'h' = snd (months l !! fromEnum mon)
       decode 'C' = show2 (year `quot` 100)
       decode 'c' = doFmt (dateTimeFmt l)
       decode 'D' = doFmt "%m/%d/%y"
       decode 'd' = show2 day
       decode 'e' = show2' day
       decode 'H' = show2 hour
       decode 'I' = show2 (to12 hour)
       decode 'j' = show3 yday
       decode 'k' = show2' hour
       decode 'l' = show2' (to12 hour)
       decode 'M' = show2 min
       decode 'm' = show2 (fromEnum mon+1)
       decode 'n' = "\n"
       decode 'p' = (if hour < 12 then fst else snd) (amPm l)
       decode 'R' = doFmt "%H:%M"
       decode 'r' = doFmt (time12Fmt l)
       decode 'T' = doFmt "%H:%M:%S"
       decode 't' = "\t"
       decode 'S' = show2 sec
       decode 's' = ...                -- Implementation-dependent
       decode 'U' = show2 ((yday + 7 - fromEnum wday) `div` 7)
       decode 'u' = show (let n = fromEnum wday in
                          if n == 0 then 7 else n)
       decode 'V' =
           let (week, days) =
                  (yday + 7 - if fromEnum wday > 0 then
                              fromEnum wday - 1 else 6) `divMod` 7
           in  show2 (if days >= 4 then
                         week+1
                      else if week == 0 then 53 else week)
       decode 'W' =
           show2 ((yday + 7 - if fromEnum wday > 0 then
                              fromEnum wday - 1 else 6) `div` 7)
```

https://doi.org/10.1017/S0956796803002612 Published online by Cambridge University Press

```
        decode 'w' = show (fromEnum wday)
        decode 'X' = doFmt (timeFmt l)
        decode 'x' = doFmt (dateFmt l)
        decode 'Y' = show year
        decode 'y' = show2 (year `rem` 100)
        decode 'Z' = tzname
        decode '%' = "%"
        decode c   = [c]

show2, show2', show3 :: Int -> String
show2 x = [intToDigit (x `quot` 10), intToDigit (x `rem` 10)]

show2' x = if x < 10 then [ ' ', intToDigit x] else show2 x

show3 x = intToDigit (x `quot` 100) : show2 (x `rem` 100)
```

https://doi.org/10.1017/S0956796803002612 Published online by Cambridge University Press