Class Time < Object

Time is an abstraction of dates and times. Time is stored internally as the number of seconds and microseconds since the *epoch*, January 1, 1970 00:00 UTC. On some operating systems, this offset is allowed to be negative. Also see the Date library module on page 742.

The Time class treats GMT (Greenwich Mean Time) and UTC (Coordinated Universal Time)[7] as equivalent. GMT is the older way of referring to these baseline times but persists in the names of calls on POSIX systems.

All times are stored with some number of microseconds. Be aware of this fact when comparing times with each other—times that are apparently equal when displayed may be different when compared.

Mixes in

Comparable:

`<, <=, ==, >=, >, between?`

Class methods

at

Time.at(*time*) → *time*
Time.at(*seconds* ⟨ , *microseconds* ⟩) → *time*

Creates a new time object with the value given by *time* or the given number of *seconds* (and optional *microseconds*) from epoch. Microseconds may be a float—this allows setting times with nanosecond granularity on systems that support it. A nonportable feature allows the offset to be negative on some systems.

```
Time.at(0)            # =>  1969-12-31 18:00:00 -0600
Time.at(946702800)    # =>  1999-12-31 23:00:00 -0600
Time.at(-284061600)   # =>  1960-12-31 00:00:00 -0600
t = Time.at(946702800, 123.456)
t.usec                # =>  123
t.nsec                # =>  123456
```

gm

Time.gm(*year* ⟨ , month ⟨ , day ⟨ , hour ⟨ , min ⟨ , sec ⟨ , usec ⟩ ⟩ ⟩ ⟩ ⟩ ⟩) → *time*
Time.gm(*sec, min, hour, day, month, year, wday, yday, isdst, tz*) → *time*

Creates a time based on given values, interpreted as UTC. The year must be specified. Other values default to the minimum value for that field (and may be nil or omitted). Months may be specified by numbers from 1 to 12 or by the three-letter English month names. Hours are specified on a 24-hour clock (0..23). Raises an ArgumentError if any values are out of range. Will also accept ten arguments in the order output by Time#to_a.

```
Time.gm(2000,"jan",1,20,15,1)   # =>  2000-01-01 20:15:01 UTC
```

7. Yes, UTC really does stand for Coordinated Universal Time. There was a committee involved.

local Time.local(*year* ⟨ , month ⟨ , day ⟨ , hour ⟨ , min ⟨ , sec ⟨ , usec ⟩ ⟩ ⟩ ⟩ ⟩ ⟩) → *time*
Time.local(*sec, min, hour, day, month, year, wday, yday, isdst, tz*) → *time*

Same as Time.gm but interprets the values in the local time zone. The second form accepts ten arguments in the order output by Time#to_a.

```
Time.local(2000,"jan",1,20,15,1)   # =>   2000-01-01 20:15:01 -0600
```

mktime Time.mktime(*year* ⟨ , month ⟨ , day ⟨ , hour ⟨ , min ⟨ , sec ⟨ , usec ⟩ ⟩ ⟩ ⟩ ⟩ ⟩) → *time*
Time.mktime(*sec, min, hour, day, month, year, wday, yday, isdst, tz*) → *time*

Synonym for Time.local.

new Time.new → *time*

Returns a Time object initialized to the current system time. **Note:** The object created will be created using the resolution available on your system clock and so may include fractional seconds.

```
a = Time.new      # =>   2009-04-13 13:26:38 -0500
b = Time.new      # =>   2009-04-13 13:26:38 -0500
a == b            # =>   false
"%.6f" % a.to_f   # =>   "1239647198.931594"
"%.6f" % b.to_f   # =>   "1239647198.932109"
```

now Time.now → *time*

Synonym for Time.new.

utc Time.utc(*year* ⟨ , month ⟨ , day ⟨ , hour ⟨ , min ⟨ , sec ⟨ , usec ⟩ ⟩ ⟩ ⟩ ⟩ ⟩) → *time*
Time.utc(*sec, min, hour, day, month, year, wday, yday, isdst, tz*) → *time*

Synonym for Time.gm.

```
Time.utc(2000,"jan",1,20,15,1)   # =>   2000-01-01 20:15:01 UTC
```

Instance methods

+ *time* + *numeric* → *time*

Addition—Adds some number of seconds (possibly fractional) to *time* and returns that value as a new time.

```
t = Time.now         # =>   2009-04-13 13:26:38 -0500
t + (60 * 60 * 24)   # =>   2009-04-14 13:26:38 -0500
```

– *time* - *time* → *float*
time - *numeric* → *time*

Difference—Returns a new time that represents the difference between two times or subtracts the given number of seconds in *numeric* from *time*.

```
t = Time.now       # =>   2009-04-13 13:26:38 -0500
t2 = t + 2592000   # =>   2009-05-13 13:26:38 -0500
t2 - t             # =>   2592000.0
t2 - 2592000       # =>   2009-04-13 13:26:38 -0500
```

<=>

time <=> *other_time* → −1, 0, +1
time <=> *other* → nil

1.9 / 1.9 Comparison—Compares *time* with *other_time* or with *numeric*, which is the number of seconds (possibly fractional) since epoch. As of Ruby 1.9, nil is returned for comparison against anything other that a Time object.

```
t = Time.now       # =>  2009-04-13 13:26:39 -0500
t2 = t + 2592000   # =>  2009-05-13 13:26:39 -0500
t <=> t2           # =>  -1
t2 <=> t           # =>  1
t <=> t            # =>  0
```

asctime

time.asctime → *string*

Returns a canonical string representation of *time*.

```
Time.now.asctime   # =>  "Mon Apr 13 13:26:39 2009"
```

ctime

time.ctime → *string*

Synonym for Time#asctime.

day

time.day → *int*

Returns the day of the month (1..*n*) for *time*.

```
t = Time.now   # =>  2009-04-13 13:26:39 -0500
t.day          # =>  13
```

dst?

time.dst? → true or false

Synonym for Time#isdst.

```
Time.local(2000, 7, 1).dst?   # =>  true
Time.local(2000, 1, 1).dst?   # =>  false
```

friday?

time.friday? → true or false

1.9 Returns true if *time*.wday is 5.

getgm

time.getgm → *time*

Returns a new Time object representing *time* in UTC.

```
t = Time.local(2000,1,1,20,15,1)  # =>  2000-01-01 20:15:01 -0600
t.gmt?                            # =>  false
y = t.getgm                       # =>  2000-01-02 02:15:01 UTC
y.gmt?                            # =>  true
t == y                            # =>  true
```

getlocal

time.getlocal → *time*

Returns a new Time object representing *time* in local time (using the local time zone in effect for this process).

```
t = Time.gm(2000,1,1,20,15,1)   # =>   2000-01-01 20:15:01 UTC
t.gmt?                          # =>   true
l = t.getlocal                  # =>   2000-01-01 14:15:01 -0600
l.gmt?                          # =>   false
t == l                          # =>   true
```

getutc

time.getutc → *time*

Synonym for Time#getgm.

gmt?

time.gmt? → true or false

Returns true if *time* represents a time in UTC.

```
t = Time.now                    # =>   2009-04-13 13:26:39 -0500
t.gmt?                          # =>   false
t = Time.gm(2000,1,1,20,15,1)   # =>   2000-01-01 20:15:01 UTC
t.gmt?                          # =>   true
```

gmtime

time.gmtime → *time*

Converts *time* to UTC, modifying the receiver.

```
t = Time.now   # =>   2009-04-13 13:26:39 -0500
t.gmt?         # =>   false
t.gmtime       # =>   2009-04-13 18:26:39 UTC
t.gmt?         # =>   true
```

gmt_offset

time.gmt_offset → int

Returns the offset in seconds between the time zone of *time* and UTC.

```
t = Time.gm(2000,1,1,20,15,1)   # =>   2000-01-01 20:15:01 UTC
t.gmt_offset                    # =>   0
l = t.getlocal                  # =>   2000-01-01 14:15:01 -0600
l.gmt_offset                    # =>   -21600
```

gmtoff

time.gmtoff → *int*

Synonym for Time#gmt_offset.

hour

time.hour → *int*

Returns the hour of the day (0..23) for *time*.

```
t = Time.now   # =>   2009-04-13 13:26:39 -0500
t.hour         # =>   13
```

isdst

time.isdst → true or false

Returns true if *time* occurs during daylight saving time in its time zone.

```
Time.local(2000, 7, 1).isdst   # =>   true
Time.local(2000, 1, 1).isdst   # =>   false
```

localtime

time.localtime → *time*

Converts *time* to local time (using the local time zone in effect for this process) modifying the receiver.

```
t = Time.gm(2000, "jan", 1, 20, 15, 1)
t.gmt?        # =>   true
t.localtime   # =>   2000-01-01 14:15:01 -0600
t.gmt?        # =>   false
```

mday

time.mday → *int*

Synonym for Time#day.

monday?

time.monday? → true or false

1.9

Returns true if *time*.wday is 1.

min

time.min → *int*

Returns the minute of the hour (0..59) for *time*.

```
t = Time.now   # =>   2009-04-13 13:26:39 -0500
t.min          # =>   26
```

mon

time.mon → *int*

Returns the month of the year (1..12) for *time*.

```
t = Time.now   # =>   2009-04-13 13:26:39 -0500
t.mon          # =>   4
```

month

time.month → *int*

Synonym for Time#mon.

nsec

time.nsec → *int*

1.9

Returns just the number of nanoseconds for *time*.

```
t = Time.now        # =>   2009-04-13 13:26:39 -0500
"%10.6f" % t.to_f   # =>   "1239647199.329925"
t.nsec              # =>   329925000
t.usec              # =>   329925
```

saturday?

time.saturday? → true or false

1.9

Returns true if *time*.wday is 06.

sec

time.sec → *int*

Returns the second of the minute $(0..60)$[8] for *time*.

```
t = Time.now   # =>  2009-04-13 13:26:39 -0500
t.sec          # =>  39
```

strftime

time.strftime(*format*) → *string*

1.9

Formats *time* according to the directives in the given format string. See Table 27.18 on the next page for the available values. Any text not listed as a directive will be passed through to the output string. If an up arrow follows the % sign, any text returned for that directive will be mapped to uppercase.

```
t = Time.now
t.strftime("Printed on %m/%d/%Y")   # =>  "Printed on 04/13/2009"
t.strftime("at %I:%M%P")            # =>  "at 01:26pm"
# force the am/pm flag to upper case
t.strftime("at %I:%M%^P")           # =>  "at 01:26PM"
```

succ

time.succ → *later_time*

1.9

Returns a time object one second after *time*.

```
now = Time.now     # =>  2009-04-13 13:26:39 -0500
later = now.succ   # =>  2009-04-13 13:26:40 -0500
# preserves the fractional part
now.to_f           # =>  1239647199.3991
later.to_f         # =>  1239647200.3991
```

sunday?

time.sunday? → true or false

1.9

Returns true if *time*.wday is 0.

thursday?

time.thursday? → true or false

1.9

Returns true if *time*.wday is 4.

to_a

time.to_a → *array*

Returns a ten-element *array* of values for *time*: [sec, min, hour, day, month, year, wday, yday, isdst, zone]. See the individual methods for an explanation of the valid ranges of each value. The ten elements can be passed directly to the methods Time.utc or Time.local to create a new Time.

```
now = Time.now   # =>  2009-04-13 13:26:39 -0500
t = now.to_a     # =>  [39, 26, 13, 13, 4, 2009, 1, 103, true, "CDT"]
```

8. Yes, seconds really can range from zero to 60. This allows the system to inject leap seconds every now and then to correct for the fact time measured by atomic clocks differs from time measured by a spinning earth.

Table 27.18. Time#strftime Directives

Format	Meaning
%a	The abbreviated weekday name (“Sun”)
%A	The full weekday name (“Sunday”)
%b	The abbreviated month name (“Jan”)
%B	The full month name (“January”)
%c	The preferred local date and time representation
%d	Day of the month (01..31)
%H	Hour of the day, 24-hour clock (00..23)
%I	Hour of the day, 12-hour clock (01..12)
%j	Day of the year (001..366)
%m	Month of the year (01..12)
%M	Minute of the hour (00..59)
%p	Meridian indicator (“AM” or “PM”)
1.9 %P	Meridian indicator (“am” or “pm”)
1.9 %s	Number of seconds since 1970-01-01 00:00:00 UTC
%S	Second of the minute (00..60)
%U	Week number of the current year, starting with the first Sunday as the first day of the first week (00..53)
%W	Week number of the current year, starting with the first Monday as the first day of the first week (00..53)
%w	Day of the week (Sunday is 0, 0..6)
%x	Preferred representation for the date alone, no time
%X	Preferred representation for the time alone, no date
%y	Year without a century (00..99)
%Y	Year with century
%Z	Time zone name
%%	Literal % character

to_f

time.to_f → *float*

Returns the value of *time* as a floating-point number of seconds since epoch.

```
t = Time.now
"%10.5f" % t.to_f   # =>   "1239647199.44315"
t.to_i              # =>   1239647199
```

to_i

time.to_i → *int*

Returns the value of *time* as an integer number of seconds since epoch.

```
t = Time.now
"%10.5f" % t.to_f   # =>   "1239647199.46455"
t.to_i              # =>   1239647199
```

to_s

time.to_s → *string*

Returns a string representing *time*. Equivalent to calling Time#strftime with a format string of %a %b %d %H:%M:%S %Z %Y.

```
Time.now.to_s   # =>   "2009-04-13 13:26:39 -0500"
```

tuesday?

time.tuesday? → true or false

1.9

Returns true if *time*.wday is 2.

tv_nsec

time.tv_nsec → *int*

1.9

Synonym for Time#nsec.

tv_sec

time.tv_sec → *int*

Synonym for Time#to_i.

tv_usec

time.tv_usec → *int*

Synonym for Time#usec.

usec

time.usec → *int*

Returns just the number of microseconds for *time*.

```
t = Time.now        # =>   2009-04-13 13:26:39 -0500
"%10.6f" % t.to_f   # =>   "1239647199.506947"
t.nsec              # =>   506947000
t.usec              # =>   506947
```

utc

time.utc → *time*

Synonym for Time#gmtime.

```
t = Time.now   # =>   2009-04-13 13:26:39 -0500
t.utc?         # =>   false
t.utc          # =>   2009-04-13 18:26:39 UTC
t.utc?         # =>   true
```

utc?

time.utc? → true or false

Returns true if *time* represents a time in UTC.

```
t = Time.now                        # =>   2009-04-13 13:26:39 -0500
t.utc?                              # =>   false
t = Time.gm(2000,"jan",1,20,15,1)   # =>   2000-01-01 20:15:01 UTC
t.utc?                              # =>   true
```

utc_offset

time.utc_offset → *int*

Synonym for Time#gmt_offset.

wednesday?

time.wednesday? → true or false

1.9

Returns true if *time*.wday is 3.

wday

time.wday → *int*

Returns an integer representing the day of the week, 0..6, with Sunday == 0.

```
t = Time.now   # =>  2009-04-13 13:26:39 -0500
t.wday         # =>  1
```

yday

time.yday → *int*

Returns an integer representing the day of the year, 1..366.

```
t = Time.now   # =>  2009-04-13 13:26:39 -0500
t.yday         # =>  103
```

year

time.year → *int*

Returns the year for *time* (including the century).

```
t = Time.now   # =>  2009-04-13 13:26:39 -0500
t.year         # =>  2009
```

zone

time.zone → *string*

Returns the name of the time zone used for *time*.

```
t = Time.gm(2000, "jan", 1, 20, 15, 1)
t.zone   # =>  "UTC"
t = Time.local(2000, "jan", 1, 20, 15, 1)
t.zone   # =>  "CST"
```