

NAME

Time::Piece - Object Oriented time objects

SYNOPSIS

```
use Time::Piece;

my $t = localtime;
print "Time is $t\n";
print "Year is ", $t->year, "\n";
```

DESCRIPTION

This module replaces the standard `localtime` and `gmtime` functions with implementations that return objects. It does so in a backwards compatible manner, so that using `localtime/gmtime` in the way documented in `perlfunc` will still return what you expect.

The module actually implements most of an interface described by Larry Wall on the `perl5-porters` mailing list here: <http://www.xray.mpe.mpg.de/mailling-lists/perl5-porters/2000-01/msg00241.html>

USAGE

After importing this module, when you use `localtime` or `gmtime` in a scalar context, rather than getting an ordinary scalar string representing the date and time, you get a `Time::Piece` object, whose stringification happens to produce the same effect as the `localtime` and `gmtime` functions. There is also a `new()` constructor provided, which is the same as `localtime()`, except when passed a `Time::Piece` object, in which case it's a copy constructor. The following methods are available on the object:

<code>\$t->sec</code>	# also available as <code>\$t->second</code>
<code>\$t->min</code>	# also available as <code>\$t->minute</code>
<code>\$t->hour</code>	# 24 hour
<code>\$t->mday</code>	# also available as <code>\$t->day_of_month</code>
<code>\$t->mon</code>	# 1 = January
<code>\$t->_mon</code>	# 0 = January
<code>\$t->monname</code>	# Feb
<code>\$t->month</code>	# same as <code>\$t->monname</code>
<code>\$t->fullmonth</code>	# February
<code>\$t->year</code>	# based at 0 (year 0 AD is, of course 1 BC)
<code>\$t->_year</code>	# year minus 1900
<code>\$t->yy</code>	# 2 digit year
<code>\$t->wday</code>	# 1 = Sunday
<code>\$t->_wday</code>	# 0 = Sunday
<code>\$t->day_of_week</code>	# 0 = Sunday
<code>\$t->weekdayname</code>	# Tue
<code>\$t->day</code>	# same as <code>weekdayname</code>
<code>\$t->fullday</code>	# Tuesday
<code>\$t->yday</code>	# also available as <code>\$t->day_of_year</code> , 0 = Jan 01
<code>\$t->isdst</code>	# also available as <code>\$t->daylight_savings</code>
<code>\$t->hms</code>	# 12:34:56
<code>\$t->hms(". ")</code>	# 12.34.56
<code>\$t->time</code>	# same as <code>\$t->hms</code>
<code>\$t->ymd</code>	# 2000-02-29
<code>\$t->date</code>	# same as <code>\$t->ymd</code>
<code>\$t->mdy</code>	# 02-29-2000
<code>\$t->mdy(" / ")</code>	# 02/29/2000

```
$t->dmy                # 29-02-2000
$t->dmy(" . ")         # 29.02.2000
$t->datetime            # 2000-02-29T12:34:56 (ISO 8601)
$t->cdate               # Tue Feb 29 12:34:56 2000
"$t"                  # same as $t->cdate

$t->epoch               # seconds since the epoch
$t->tzoffset            # timezone offset in a Time::Seconds object

$t->julian_day           # number of days since Julian period began
$t->mjd                 # modified Julian date (JD-2400000.5 days)

$t->week                # week number (ISO 8601)

$t->is_leap_year        # true if it is
$t->month_last_day      # 28-31

$t->time_separator($s) # set the default separator (default ":")
$t->date_separator($s) # set the default separator (default "-")
$t->day_list(@days)    # set the default weekdays
$t->mon_list(@months)   # set the default months

$t->strftime(FORMAT)    # same as POSIX::strftime (without the overhead
                        # of the full POSIX extension)
$t->strftime()          # "Tue, 29 Feb 2000 12:34:56 GMT"

Time::Piece->strftime(String, FORMAT)
                        # see strftime man page. Creates a new
                        # Time::Piece object
```

Local Locales

Both `wdayname` (day) and `monname` (month) allow passing in a list to use to index the name of the days against. This can be useful if you need to implement some form of localisation without actually installing or using locales.

```
my @days = qw( Dimanche Lundi Merdi Mercredi Jeudi Vendredi Samedi );
```

```
my $french_day = localtime->day(@days);
```

These settings can be overridden globally too:

```
Time::Piece::day_list(@days);
```

Or for months:

```
Time::Piece::mon_list(@months);
```

And locally for months:

```
print localtime->month(@months);
```

Date Calculations

It's possible to use simple addition and subtraction of objects:

```
use Time::Seconds;

my $seconds = $t1 - $t2;
$t1 += ONE_DAY; # add 1 day (constant from Time::Seconds)
```

The following are valid (\$t1 and \$t2 are Time::Piece objects):

```
$t1 - $t2; # returns Time::Seconds object
$t1 - 42;  # returns Time::Piece object
$t1 + 533; # returns Time::Piece object
```

However adding a Time::Piece object to another Time::Piece object will cause a runtime error.

Note that the first of the above returns a Time::Seconds object, so while examining the object will print the number of seconds (because of the overloading), you can also get the number of minutes, hours, days, weeks and years in that delta, using the Time::Seconds API.

Date Comparisons

Date comparisons are also possible, using the full suite of "<", ">", "<=", ">=", "<=>", "==" and "!=".

Date Parsing

Time::Piece links to your C library's `strptime()` function, allowing you incredibly flexible date parsing routines. For example:

```
my $t = Time::Piece->strptime("Sun 3rd Nov, 1943",
                              "%A %drd %b, %Y");

print $t->strftime("%a, %d %b %Y");
```

Outputs:

```
Wed, 03 Nov 1943
```

(see, it's even smart enough to fix my obvious date bug)

For more information see "man strptime", which should be on all unix systems.

YYYY-MM-DDThh:mm:ss

The ISO 8601 standard defines the date format to be YYYY-MM-DD, and the time format to be hh:mm:ss (24 hour clock), and if combined, they should be concatenated with date first and with a capital 'T' in front of the time.

Week Number

The *week number* may be an unknown concept to some readers. The ISO 8601 standard defines that weeks begin on a Monday and week 1 of the year is the week that includes both January 4th and the first Thursday of the year. In other words, if the first Monday of January is the 2nd, 3rd, or 4th, the preceding days of the January are part of the last week of the preceding year. Week numbers range from 1 to 53.

Global Overriding

Finally, it's possible to override `localtime` and `gmtime` everywhere, by including the 'override' tag in the import list:

```
use Time::Piece ':override';
```

AUTHOR

Matt Sergeant, matt@sergeant.org Jarkko Hietaniemi, jhi@iki.fi (while creating Time::Piece for core perl)

License

This module is free software, you may distribute it under the same terms as Perl.

SEE ALSO

The excellent Calendar FAQ at <http://www.tondering.dk/claus/calendar.html>

BUGS

The test harness leaves much to be desired. Patches welcome.