NAME

Date::Manip::Date − Methods for working with dates

SYNOPSIS

```
use Date::Manip::Date;
$date = new Date::Manip::Date;
```

DESCRIPTION

This module works specifically with date objects.

Although the word date is used extensively here, it is actually somewhat misleading. Date::Manip works with the full calendar date (year, month, day, and week when appropriate), time of day (hour, minute, second), and time zone. It doesn't work with fractional seconds.

METHODS

base
config
err
is_date
is_delta
is_recur
new
new_config
new_date
new_delta
new_recur
tz Please refer to the Date::Manip::Obj documentation for these methods.

calc

```
$date2 = $date−>calc($delta [,$subtract]);
$delta = $date−>calc($date2 [,$subtract] [,$mode]);
```

Please refer to the Date::Manip::Calc documentation for details.

cmp

```
$val = $date1−>cmp($date2);
```

This compares two different dates (both of which must be valid date objects). It returns −1, 0, or 1 similar to the cmp or <=> operators in perl. The comparison will automatically handle time zone differences between the two dates (i.e. they will be sorted in order as they appear in the GMT zone).

A warning is printed if either of the date objects does not include a valid date.

complete

```
$flag = $date−>complete([$field]);
```

This tests the date stored in the object to see if it is complete or truncated (see below for a discussion of this).

If no `$field` is passed in, it returns 1 if the date is complete, or 0 if it was truncated and default values have been supplied.

If `$field` is passed in, it may be one of: m, d, h, mn, s . It will return 1 if the value for that field was specified, or 0 if a default was used.

convert

```
$err = $date−>convert([$zone]);
```

This converts the date stored in the object to a different time zone. `$zone` can be the name of a time zone. If it is not passed in, the date is converted to the local time zone.

holiday

```
$name = $date->holiday();
@name = $date->holiday();
$name = $date->event();
```

This returns the name of the holiday if `$date` is a holiday. If `$date` is not a holiday, undef is returned. If `$date` is an unnamed holiday, an empty string is returned.

In scalar context, holiday returns the name of one holiday that occurs on that date (the one first defined in the config file). In list context, it returns all holidays on that date.

input

```
$str = $date->input();
```

This returns the string that was parsed to form the date.

is_business_day

```
$flag = $date->is_business_day($checktime);
```

This returns 1 if `$date` is a business day.

`$checktime` may be passed in. If it is non-zero, the time is checked to see if the date is a business day and falls within work hours.

list_holidays

```
 @date = $date->list_holidays([$y]);
```

This returns a list of Date::Manip::Date objects containing all dates during a year which are holidays. The times will all be 00:00:00.

If `$y` is not passed in, it will list the holidays in the same year as the date stored in `$date` (if any) or in the current year otherwise.

list_events

```
@list = $date->list_events(       [$format] );
@list = $date->list_events(0      [,$format]);
@list = $date->list_events($date1 [,$format]);
```

This returns a list of events. Events are defined in the Events section of the config file (discussed in the Date::Manip::Holidays manual).

In the first form, a list of all events active at the precise time stored in `$date` will be returned.

If the first argument evaluates to 0, a list of all events active at any time during that day (Y,M,D) are returned.

If the first argument is another date object, all events that are active at any time between the two dates (inclusive) are returned.

By default, the list returned is of the form:

```
( [START, END, NAME],
  [START, END, NAME],
  ...
)
```

where START is a date object when an event starts, END is a date object when it ends, and NAME is the name of the event. Note that START and END are the actual start and end date of the event and may be outside the range of dates being examined (though the event will obviously overlap the range or it wouldn’t be included in the list).

If `$format` is included, it can specify an alternate format for the output. Currently, the only supported format is named ‘‘dates’’ and it returns a list in the form:

```
( [DATE1, NAME1a, NAME1b, ...],
  [DATE2, NAME2a, NAME2b, ...],
  ...
)
```

This includes a list of all dates during the range when there is a change in what events are active. DATE1 will always be the start of the range being considered, and (NAME1a, NAME1b, ...) are the list of all events that will be active at that time. At DATE2, the list of active events changes with (NAME2a, NAME2b, ...) being active.

It is quite possible that a date be included which has no active events, and in that case, the list of names will be empty.

nearest_business_day

```
$date−>nearest_business_day([$tomorrowfirst]);
```

This looks for the work day nearest to `$date`. If`$date` is a w ork day, it is left unmodified. Otherwise, it will look forward or backwards in time 1 day at a time until a work day is found. If `$tomorrowfirst` is non-zero (or if it is omitted and the config variable TomorrowFirst is non-zero), we look to the future first. Otherwise, we look in the past first. In other words, in a normal week, if `$date` is Wednesday, `$date` is returned. If `$date` is Saturday, Friday is returned. If `$date` is Sunday, Monday is returned. If Wednesday is a holiday, Thursday is returned if `$tomorrowfirst` is non-nil or Tuesday otherwise.

next_business_day
prev_business_day

```
$date−>next_business_day($off [,$checktime]);
$date−>prev_business_day($off [,$checktime]);
```

The next_business_day method sets the given date to `$off` (which can be a positive integer or zero) business days in the future. The prev_business_day method sets the date to `$off` business days in the past.

First, `$date` is tested. If `$checktime` is nonzero, the date must fall on a business date, and during business hours. If `$checktime` is zero, the time check is not done, and the date must simply fall on a business date.

If the check fails, the date is moved to the start of the next business day (if `$checktime` is nonzero) or the next business day at the current time (if `$checktime` is zero). Otherwise, it is left unmodified.

Next, if `$off` is greater than 0, the day `$off` work days from now is determined.

One thing to note for the prev_business_day method is that if `$date` check fails, the date is set to the next business date, exactly like next_business_day. In other words, if `$date` is not a business day, the call:

```
$date−>prev_business_day(0 [,$checktime]);
```

moves `$date` forward in time instead of backward which is nonintuitive, but you just have to think of day 0 as being the next business day if `$date` is not a business day.

As a result, the following two calls ALWAYS give the same result:

```
$date−>next_business_day(0 [,$checktime]);
$date−>prev_business_day(0 [,$checktime]);
```

no matter what date is stored in `$date`.

parse

```
$err = $date−>parse($string [,@opts]);
```

This parses a string which should include a valid date and stores it in the object. If the string does not include a valid date, an error is returned. Use the err method to see the full error message.

A full date may include a calendar date (year, month, day), a time of day (hour, minute, second), and time zone information. All of this can be entered in many different formats.

For information on valid date formats, refer to the section VALID DATE FORMATS. For information on valid time zone information, refer to the section VALID TIME ZONE FORMATS.

If no time zone information is included in the date, it is treated as being in the local time zone.

If time zone information is included, the date will be kept in that time zone, and all operations will be done in that time zone. The convert method can be used to change the time zone to the local time zone, or to another time zone.

Some things to note:

All strings are case insensitive. ‘‘December’’ and ‘‘DEceMBer’’ are equivalent.

When a part of the date is not given, defaults are used. This is described below in the section ‘‘Complete vs. truncated dates and times’’.

The year may be entered as 2 or 4 digits. If entered as 2 digits, it will be converted to a 4 digit year. There are several ways to do this based on the value of the YYtoYYYY config variable. Refer to the Date::Manip::Config documentation for more details.

Dates are always checked to make sure they are valid.

If any other arguments are passed in, they act as options which may improve the speed of parsing. These include:

```
   noiso8601  Do not try to parse the
              date as an ISO 8601 date
              or time.
   nodow      Do not try to parse a
              day−of−week (Monday) in
              the string.
   nocommon   Do not try to parse the
              date using the formats
              in the "Common date
              formats" section.
   noother    Do not try to parse the
              date using the "Less common
              date formats" or a time
              using the "Other time
              formats".
   nospecial  Do not try to parse the
              date using the "Special
              date strings" formats
              or a time using the
              "Special time strings"
              formats, or as a
              combined date/time using
              the "Additional combined
              date and time" formats.
   nodelta    Do not treat deltas as
              a date relative to now.
   noholidays Do not parse holiday
              names as dates.
```

parse_date

```
   $err = $date−>parse_date($string [,@opts]);
```

This parses a string which contains a valid date and sets the date part of the object.

If the object contained a valid date, the time is kept unchanged. If the object did NOT contain a valid date, a time of 00:00:00 is used.

`@opts` can be any of the strings described in the parse method above.

parse_time

```
$err = $date−>parse_time($string [,@opts]);
```

This parses a string and sets the time portion of `$date` to contain it.

If the object contained a valid date, the Y/M/D portion is left unchanged. Otherwise, the current date is used.

`@opts` can be 'noiso8601' or 'noother'.

parse_format

```
$err          = $date−>parse_format($format,$string);
($err,%match) = $date−>parse_format($format,$string);
```

This will parse a date contained in `$string` based on explicit format information contained in `$format`.

If the format is invalid, `$err` will contain an error message. If the format is valid, but string doesn't match, an error code of 1 is returned.

If called in array context, a hash will be returned containing %+. This is primarily useful if the `$format` string contains some named capture groups that you define. This is discussed below.

`$format` is a string containing a regular expression with some special directives (based on the printf directives). These directives are turned into regular expression components, and then the entire string is turned into a regular expression which, if `$string` matches it, will return the date.

The directives available are identical to the printf directives. So, if your `$format` string contains the directive '%Y', it will match a 4−digit year.

All of the printf directives are available here with a few caveats:

```
%l        This directive is NOT available.

%b,%h,%B  These will all match a month name or abbreviation.

%v,%a,%A  These will all match a day name or abbreviation.

%z,%Z,%N  These will match any time zone string.

%n        Multi−line matching is not currently supported,
          so this directive is not allowed.

%x        All format directives are converted to a regular
          expression and then cached (so that a format
          can be reused without the penalty of doing the
          conversion to a regular expression with each use).
          As a result, if you need to set the DateFormat config
          variable (which determines the meaning of the %x
          directive), it must be done before a format string
          containing %x is used. If the DateFormat config variable
          is set afterwards, the format string will reflect the
          old, NOT THE NEW, value of DateFormat.
```

The format string may not over-specify the date. In other words, you may not include both a `%y` and `%Y` directive or both a `%j` and `%m` directive.

A valid format string will specify any of the following sets of data:

```
Required          Optional

M D H Mn S        Y Zone Day−of−week
M D H Mn          Y Zone Day−of−week
M D               Y Zone Day−of−week
H Mn S            Zone
H Mn              Zone
```

For example, if you had a date stored as:

```
YYYY.MM−DD
```

you could match it using the following:

```
$date−>parse_format('%Y\\.%m\\−%d',$string);
```

If you wanted to extract the date from an apache log line:

```
10.11.12.13 − − [17/Aug/2009:12:33:30 −0400] "GET /favicon.ico ...
```

you could use:

```
$date−>parse_format('.*?\\[%d/%b/%Y:%T %z\\].*',$line);
```

When matching months, days, and hours, there are two directives that could be used (for numerical versions). For the month, you may use `%m` or `%f`. If your date is known to have a two-digit month, you should use `%m`. If it contains a one− or two-digit month, you must use `%f` (and it is safe to use `%f` for two-digit months). Similarly, for days, you can use `%d` or `%e` and for hours you can use `%H` or `%k`. In both cases, the first can only be used if you are guaranteed a 2−digit value.

In your format string, you may use capture groups (or back references to them) in the regular expression using all of the rules of normal regular expressions. Since Date::Manip uses named capture groups internally, it is suggested that you also use named groups. Mixing numbered and named groups will work... but it'll be entirely up to you to keep track of what numbers refer to which capture groups.

Every printf directive adds one or more named capture groups to the regular expression. If you use named groups in the format string, they must not conflict with the ones used internally, or else the date will probably not be parsed correctly.

The following named capture groups are used internally:

```
y
m
d
h
mn
s
mon_name
mon_abb
dow_name
dow_abb
dow_char
dow_num
doy
nth
ampm
epochs
epocho
```

```
   tzstring
   off
   abb
   zone
   g
   w
   l
   u
```

To be safe, it is suggested that any additional named capture groups introduced by the programmer start with a capital letter. This is guaranteed to never conflict with any existing, or future named capture groups.

In order to get access to the values stored in the additional named capture groups, the parse_format function must be called in list context, and the %+ array will be returned as the second value.

As an example:

```
   $string = "before 2014−01−25 after";
   ($err,%m) = $date−>parse_format('(?<PRE>.*?)%Y−%m−%d(?<POST>.*)',$string);
```

would return a hash (%m) with the following key/value pairs:

```
   'PRE'  => 'before '
   'POST' => ' after'
```

prev
next

The prev method changes the date to the previous (or current) occurrence of either a day of the week, a certain time of day, or both. The next method changes the date to the next (or current) occurrence. The examples below illustrate the prev method, but the next one is identical in operation.

There are two different ways to use this method. The first is to pass in a day of week and possibly a time:

```
   $err = $date−>prev($dow, $curr [,$time]);
```

If `$curr` = 0, this means to look for the previous occurrence of the day of week, and set the time to the value passed in (or current time if no time was passed in). The day is ALWAYS less than the current day. If the current day is the same day of week as `$dow`, then the date returned will be one week earlier.

If `$curr` = 1, it means to look for the current or previous occurrence of the day of week, and set the time to the value passed in (or 00:00:00 if none was passed in). If the current day of week is the same as `$dow`, the date will remain unchanged. Since the time is then set, the new date may actually occur after the original date depending on the value of `$time`.

If `$curr` = 2, it means to look for the last time (not counting now) that the day of week at the given time occurred. The date may be the same as the original date.

`$time` may be a list reference of [H,MN,S], [H,MN], or [H].

The following examples should illustrate the use of this function.

```
    Original Date = Fri Nov 22 18:15:00

    dow      curr   time       new date

    4 (Thu)  0/1/2  undef      Thu Nov 21 00:00:00
    4        0/1/2  [12,30,0]  Thu Nov 21 12:30:00

    5 (Fri)  0/2    undef      Fri Nov 15 18:15:00
    5        1      undef      Fri Nov 22 18:15:00

    5        0      [12,30,0]  Fri Nov 15 12:30:00
    5        1/2    [12,30,0]  Fri Nov 22 12:30:00

    5        0/2    [19,30,0]  Fri Nov 15 19:30:00
    5        1      [19,30,0]  Fri Nov 22 19:30:00
```

The second way to use this method is by passing in undef for the day of week.

```
   $err = $date->prev(undef,$curr,$time);
```

In this case, a time is required and it must be a list reference of 3 elements: [H, MN, S]. Any or all of the elements may be undef.

The new date is the previous occurrence of the time.

If you define hours, then minutes and seconds may be defined, or default to 0 and you are looking for a previous time that the specified time (HH:00:00) occurred (which might be as much as 24 hours in the past).

If hours are undefined and minutes are defined, then seconds may be defined, or default to 0, and you are looking for the last time the minutes/seconds (MN:SS) appeared on the digital clock, which will be sometime in the past hour.

Finally, if hours and minutes are undefined, seconds must be defined (or default to zero) and the last time that that second occurred will be returned (which will be sometime in the past minute).

If `$curr` is non-zero, the current time is returned if it matches the criteria passed in, so the returned value will be now or in the past. If `$curr` is zero, the time returned will definitely be in the past.

```
    DATE = Fri Nov 22 18:15:00

    curr  hr     min    sec      returns
    0/1   18     undef  undef    Nov 22 18:00:00
    0/1   18     30     0        Nov 21 18:30:00
    0     18     15     undef    Nov 21 18:15:00
    1     18     15     undef    Nov 22 18:15:00
    0     undef  15     undef    Nov 22 17:15:00
    1     undef  15     undef    Nov 22 18:15:00
```

printf

```
   $out = $date->printf($in);
   @out = $date->printf(@in);
```

This takes a string or list of strings which may contain any number of special formatting directives. These directives are replaced with information contained in the date. Everything else in the string is returned unmodified.

A directive always begins with '%'. They are described in the section below in the section PRINTF DIRECTIVES.

secs_since_1970_GMT

```
   $secs = $date->secs_since_1970_GMT();
```

This returns the number of seconds that have elapsed since Jan 1, 1970 00:00:00 GMT (negative if the date is earlier).

The reverse is also allowed:

```
   $err = $date->secs_since_1970_GMT($secs);
```

which sets the date to `$secs` seconds from Jan 1, 1970 00:00:00 GMT in the local time zone.

set

```
$err = $date−>set($field,@vals [,$isdst]);
```

This explicitly sets one or more fields in a date.

`$field` can be any of the following:

```
$field   @vals

zone     [ZONE]         ZONE can be any zone or alias

zdate    [ZONE,]DATE    sets the zone and entire date

date     DATE           sets the entire date

time     TIME           sets the entire time

y        YEAR           sets one field
m        MONTH
d        DAY
h        HOUR
mn       MINUTE
s        SECOND
```

Here, DATE is a list reference containing [Y,M,D,H,MN,S] and TIME is a list reference containing [H,MN,S].

ZONE is optional (it defaults to the local zone as defined either by the system clock, or the SetDate or ForceDate config variables). If it is passed in, it can be any zone name, abbreviation, or offset. An offset can be expressed either as a valid offset string, or as a list reference. Refer to the join/split functions of Date::Manip::Base for information on valid offset strings.

An optional last argument is `$isdst` (which must be 0 or 1) is included when setting a date which could be in either standard time or daylight saving time. It is ignored in all other situations. If it is not included, and the resulting date could be in either, it will default to standard time.

The `$date` object must contain a valid date (unless the entire date is being set with `$field` set to either ‘‘zdate’’ or ‘‘date’’).

If `$field` is ‘‘zone’’, the time zone of the date will be set. If ZONE is not passed in, it will be set to the local time zone. When setting the time zone, no conversion is done! Whatever date and time is stored in the `$date` object prior to this remains unchanged... except it will be that date and time in the new time zone.

If `$field` is ‘‘zdate’’, the entire date and time zone is set. If ZONE is not passed in, it is set to the local time zone.

If `$field` is ‘‘date’’, the entire date will be set, but the time zone of the date will not be changed.

If `$field` is ‘‘time’’, or one of the individual fields, only those fields will be modified.

An error is returned if an invalid argument list is passed in, or if the resulting date is checked and found to be invalid.

value

```
$val = $date−>value([$type]);
@val = $date−>value([$type]);
```

These return the value of the date stored in the object.

In scalar context, a printable string in the form YYYYMMDDHH:MN:SS is returned. In list context, a list is returned of (Y,M,D,H,MN,S).

If `$type` is omitted, the date is returned in the time zone it was parsed in.

If `$type` is "local", it is returned in the local time zone (which is either the system time zone, or the zone specified with the SetDate or ForceDate config variables).

If `$type` is "gmt", the date is returned in the GMT time zone.

An empty string or list is returned in the case of an error (and an error code is set).

week_of_year

```
   $wkno = $date->week_of_year([$first]);
```

This figures out the week number. If `$first` is passed in, it must be between 1 and 7 and refers to the first day of the week. If `$first` is not passed in, the FirstDay config variable is used.

NOTE: This routine should only be called in rare cases. Use printf with the `%W`, `%U`, `%J`, `%L` formats instead. This routine returns a week between 0 and 53 which must then be "fixed" to get into the ISO 8601 weeks from 1 to 53. A date which returns a week of 0 actually belongs to the last week of the previous year. A date which returns a week of 53 may belong to the first week of the next year.

ISSUES WITH PARSING DATES

The following issues may occur when parsing dates that should be understood to make full use of this module.

Complete vs. truncated dates and times

Date formats are either complete or truncated. A complete date fully specifies the year, month, and day and a complete time fully specifies the hour, minute, and second.

It should be understood that in many instances, the information may be implied rather than explicitly stated, but it is still treated as complete.

For example, the date "January 3" is complete because it implies the current year.

A truncated calendar date or time does not include information about some of the fields. Date::Manip will never work with a partial date or time, so defaults will be supplied.

For example, the date "2009−01" is missing a day field, so a default will be used. In this case, the day will be the 1st, so this is equivalent to "Jan 1st 2009". If only the year is given, it will default to Jan 1.

If the time, or any of it's components is missing, they default to 00. So the time "12:30" and "12:30:00" are equivalent.

The "complete" method can be used to check what type of date was parsed, and which values were specified (either explicitly or implied) and which were provided as a default. It should be noted that there is no way to differentiate between an explicit and implied value.

A string with a date and/or time may consist of any of the following:

```
   a complete date and a time (complete or truncated)
   a truncated date with no time
   a time (complete or truncated) with no date
```

In other words, the date "Jan 2009 12:30" is not valid since it consists of a time with a truncated date.

VALID TIME ZONE FORMATS

When specifying a time zone, it can be done in three different ways. One way is to specify the actual time zone. The second is to supply a valid time zone abbreviation. The third is to specify an offset (with an optional abbreviation). The following dates illustrate the these formats.

The timezone information always follows the time immediately, and may only be included if a time is included. The following examples use an ISO 8601 format for the date/time, but any of the other date and time formats may be used.

The first way to specify the time zone is to specify it by complete name (or using one of the standard aliases):

```
2001-07-01-00:00:00 America/New_York
```

Although this is unambiguous when it comes to determining the time zone, the time is ambiguous in most zones for one hour of the year. When a time change occurs during which the clock is moved back, the same wall clock time occurs twice.

For example, in America/New_York, on Sunday, Nov 2, 2008, at 02:00 in the morning, the clock was set back to 01:00. As a result, the date Nov 2, 2008 at 01:30 is ambiguous. It is impossible to determine if this refers to the 01:30 that occurred half an hour before the time change, or the one 30 minute after the change.

In practice, if this form is used, the date will be assigned to standard time, meaning that there will be some times (typically 1 hour per year) which cannot be expressed this way. As such, this method is discouraged.

The second way to specify the time zone, which is the most common, is to use a time zone abbreviation:

```
2001-07-01-00:00:00 EDT
```

Unfortunately, the abbreviation does not uniquely determine the time zone except in a few cases. In order to assign a time zone, Date::Manip will refer to a list of all time zones which use the abbreviation. They will be tested, in the order given in the Date::Manip::Zones documentation, and the first match (i.e. the one in which the given date/time and abbreviation are valid) determines the time zone which will be used. A great deal of effort has been made to ensure that the most likely time zone will be obtained (i.e. the most common time zones are tested before less common ones), so in most cases, the desired results will be obtained.

If the default order does not yield the desired time zone, the order of testing can be modified using the abbrev method described in the Date::Manip::TZ documentation.

Although the time zone is ambiguous, the date is not, since only time zones for which the date are valid will be used.

The third way to specify the time zone is by specifying an offset and an optional abbreviation:

```
2001-07-01-00:00:00 -04
2001-07-01-00:00:00 -0400
2001-07-01-00:00:00 -040000
2001-07-01-00:00:00 -04:00
2001-07-01-00:00:00 -04:00:00

2001-07-01-00:00:00 -04 (EDT)
2001-07-01-00:00:00 -0400 (EDT)
2001-07-01-00:00:00 -040000 (EDT)
2001-07-01-00:00:00 -04:00 (EDT)
2001-07-01-00:00:00 -04:00:00 (EDT)

2001-07-01-00:00:00 -04 EDT
2001-07-01-00:00:00 -0400 EDT
2001-07-01-00:00:00 -040000 EDT
2001-07-01-00:00:00 -04:00 EDT
2001-07-01-00:00:00 -04:00:00 EDT
```

The offset almost never sufficient to uniquely determine the time zone (and it is not even guaranteed that both the offset and abbreviation will, though in practice, it is probably sufficient). In this instance, the time zone will be determined by testing all time zones which have the given offset (and abbreviation if it is included) until one is found which matches both pieces of information. For more information about how this testing is done, refer to the def_zone method of the Date::Manip::TZ documentation.

VALID DATE FORMATS

There are several categories of date formats supported by Date::Manip. These are strings which specify only the year/month/day fields.

These formats explicitly set the date, but not the time. These formats may be combined with a time string (as specified below) to set both the date and time. If this is not done, the default time is determined by the DefaultTime config variable.

ISO 8601 dates

The preferred date formats are those specified by ISO 8601. The specification includes valid calendar date and valid time formats. Date::Manip will handle all of these formats, but does not require that the dates rigidly adhere to the specification since the ultimate goal of Date::Manip is to handle dates as they are represented in real life and some common variations exist which are similar to, but not identical to, those from the specification.

A calendar date includes the following fields:

```
CC    2−digit representation of the century
YY    2−digit representation of the year in
      a century
MM    2−digit representation of a month
DD    2−digit representation of a day of month
DoY   3−digit representation of a day of year
      (001−366)
Www   the character "W" followed by a 2−digit
      week of the year (01−53)
D     the day of the week (1−7)
```

The following date formats are considered complete by Date::Manip. In the following, the date Thu Mar 5 2009 is used as an example. This is the 64th day of the year. Thu is the 4th day of the week. The week starting Mon, Mar 2 is the 10th week of the year (according the the ISO 8601 definition). Obviously, some of the formats are only valid when used at some times. For example, the format −−MMDD refers to a month and day in the current year, so the date Mar 5, 2009 can only be specified using this format during 2009.

```
Format      Notes   Examples

CCYYMMDD            20090305
CCYY−MM−DD          2009−03−05

YYMMDD      1,2,4   090305
YY−MM−DD            09−03−05

−YYMMDD     3,4     −090305
−YY−MM−DD           −09−03−05

−−MMDD      1       −−0305
−−MM−DD             −−03−05

−−−DD       1       −−−05


CCYYDoY             2009064
CCYY−DoY            2009−064

YYDoY       1,4     09064
YY−DoY              09−064

−YYDoY      3,4     −09064
−YY−DoY             −09−064

−DoY        1       −064
```

```
   CCYYWwwD                2009W104
   CCYY-Www-D              2009-W10-4

   YYWwwD      1,4         09W104
   YY-Www-D                09-W10-4

   -YYWwwD     3,4         -09W104
   -YY-Www-D               -09-W10-4

   -YWwwD      1           -9W104
   -Y-Www-D                -9-W10-4
                           Y is the year (0-9) in
                           current decade

   -WwwD       1           -W104
   -Www-D                  -W10-4

   -W-D        1           -W-4
                           D is day (1-7) in
                           current week

   ---D        1           ---4
                           same as -W-D
```

The following date formats are truncated:

```
   CCYY-MM     2           2009-03   (2009-03-01)

   CCYY                    2009      (2009-01-01)

   CC          2           20        (2000-01-01)

   -YYMM       4           -0903
   -YY-MM                  -09-03

   -YY         4           -09

   --MM                    --03

   CCYYWww                 2009W10
   CCYY-Www                2009-W10

   YYWww       4           09W10
   YY-Www                  09-W10

   -YYWww      3,4         -09W10
   -YY-Www                 -09-W10

   -Www                    -W10
```

Notes:

1 These formats are considered truncated in the standard, but since they do include (or imply, using the current date for defaults)

all of the fields, and since they do not introduce any parsing
complexities, the standard is relaxed, and they are treated as
complete.

2 These formats are treated differently than in Date::Manip 5.xx as
described below.

3 These formats are not defined in the ISO 8601 spec, but
are added for the sake of completeness since they do not
add any parsing incompatibilities.

4 Formats where the century is not given are described as a year in
the current century in the specification. Date::Manip treats this
more generically using the YYtoYYYY config variable. This will be
used to determine how to determine the full year.

Date::Manip 5.xx handled ISO 8601 dates in a less rigid fashion, and deviated from the specification in several formats. As of 6.00, the specification is followed much more closely so that all of the date formats included in it should produce valid dates. This changes, in a backwards incompatible way, the way a few strings will be interpreted as dates.

As of 6.00, a two-digit date will be treated as CC. Previously, it was treated as YY.

A six-digit date will be treated as YYMMDD. Previously, it was treated as YYYYMM.

Previously, dashes were treated as optional in many cases. According to the specification, dates may be written in expanded form (with all dashes present) or abbreviate form (with no dashes). As of 6.00, this is the behavior, so the formats: YYMMDD and YY-MM-DD are allowed, as per the specification, but the format YY-MMDD is NOT allowed (though it was previously).

The Www-D formats require a bit of explanation. According to the specification, the date:

```
   1996−w02−3
```

refers to the day with an ordinal number of 3 within the calendar week in the 2nd week of 1996.

In the specification, the days of the week are numbered from 1 to 7 (Monday to Sunday), and the week always begins on Monday, so day 1 (Monday) is always the first day of the week, day 2 (Tuesday) is always the second day of the week, etc.

In Date::Manip, the constraint that the week must start with Monday is relaxed, allowing the week to begin with Sunday (a far more common start of the week in calendars, at least in some parts of the world).

This presents a problem though in that the above date could be interpreted as Wednesday (day 3) of the 2nd week of 1996, or as the 3rd day of the 2nd week of 1996 (which would normally be Wednesday, but would be Tuesday if the week begins on Sunday).

As of Date::Manip 6.00, the above date will be interpreted as the 3rd day of the 2nd week. This is a reversal from Date::Manip 5.xx, but I believe is what the specification would require. For more information, refer to the Date::Manip::Changes document.

Common date formats

Date::Manip supports a number of common date formats. The following fields may be included in a date:

```
YY    2−digit representation of the year
YYYY  4−digit representation of the year
M     1− or 2− digit representation of the month
MM    2−digit representation of the month
D     1− or 2− digit representation of the day
DD    2−digit representation of the day
mmm   The abbreviated or full month name (i.e. Jan)
```

The following date formats are supported:

```
Format       Notes    Examples

M/D          1,2,3    3/5
M/D/YY       1        3/5/09
M/D/YYYY     1        3/5/2009

YYYY/M/D              2009/3/5

mmm/D                 Mar/5
mmm/D/YY              Mar/5/09
mmm/D/YYYY            Mar/5/2009
D/mmm                 5/Mar
D/mmm/YY              5/Mar/09
D/mmm/YYYY            5/Mar/2009
YYYY/mmm/D            2009/Mar/5

mmmD                  Mar5
mmmDDYY      4        Mar0509
mmmDDYYYY             Mar052009
Dmmm                  5Mar
DmmmYY                5Mar09
DmmmYYYY              5Mar2009
YYYYmmmD              2009Mar5

mmmD YY               Mar5 09
mmmD YYYY             Mar5 2009
Dmmm YY               5Mar 09
Dmmm YYYY             5Mar 2009

mmm/D YY              Mar/5 09
mmm/D YYYY            Mar/5 2009
D/mmm YY              5/Mar 09
D/mmm YYYY            5/Mar 2009

YY   mmmD             09   Mar5
YYYY mmmD             2009 Mar5
YY   Dmmm             09   5Mar
YYYY Dmmm             2009 5Mar

YY   mmm/D            09   Mar/5
YYYY mmm/D            2009 Mar/5
YY   D/mmm            09   5/Mar
YYYY D/mmm            2009 5/Mar

YYYY:MM:DD            2010:01:15 (EXIF format)
```

```
mmmYYYY     4       Jun 2010
YYYYmmm     4       2010 June
mmm/YYYY    4       Jun/2010
YYYY/mmm    4       2010/Jun
```

In the formats above, the slash (/) can be replace by any of the valid separators: whitespace, slash (/), period (.), or dash (−). The dash, though allowed, is discouraged since it may conflict with an ISO 8601 format. For example, the format MM/DD/YY is just fine, but MM-DD-YY does not work since it conflicts with YY-MM-DD. To be safe, if ''−'' is used as a separator in a non-ISO format, they should be turned into ''/'' before calling the Date::Manip routines or you should use the 'noiso8601' option with the **parse** or **parse_date** methods.

No matter what separator is used, the same separator must be used throughout the date. For example, MM/DD/YY is valid and MM.DD.YY is also valid, but MM/DD.YY is NOT valid.

Notes:

1 With these formats, Americans tend to write month first, but many
  other countries tend to write day first. The latter behavior can be
  obtained by setting the config variable DateFormat to something other
  than ''US''.

2 The dot (.) separator may not be used in the M/D format since it
  will be interpreted as the H12,H+ format described below.

3 The M/D format should not use the period (.) separator as that will
  incorrectly match the HH,H+ time format.

4 Historically, I have not supported partial dates (i.e. dates that
  were not fully specified), but it has been argued that something like
  'Jun 1910' would be interpreted by almost everyone as a day in June
  of 1910 instead of June 19, 2010. And it has been shown that in
  some applications, dates are specified in that way. I have added the
  new config variable Format_MMMYYYY which will change this. If this
  variable is not set, the formats allowed are:

```
   mmmDDYY

If it is set, the formats allowed are:

   mmmYYYY
   YYYYmmm

The day of week may not be included with these formats.  When
parsing a full date/time, if Format_MMMYYYY is set to 'first',
it returns the 1st of the month at midnight.  If it is set to
'last', it returns the last day at 23:59:59.  If parsing only
only a date, it will be set to the first or last day of the
month at midnight.
```

These formats explicitly set the date, but not the time. The default time is determined by the DefaultTime config variable.

Less common date formats

The following formats are also supported by Date::Manip:

```
DoW
     The day of week of the current week
        Friday
        Friday at 12:40

MMM Nth [YYYY]
Nth MMM [YYYY]
YYYY MMM Nth
YYYY Nth MMM
     Dec 1st 1970
     1st Dec 1970
     1970 Dec 1st
     1970 1st Dec

next/prev DoW
     The next or last occurrence of DoW
        next Friday
        last Friday at 12:40

next/last week/month/year
     The day one week/month/year from now
     or in the past
        next week
        last month at 15:00

last day in MMM [YYYY]
     The last day of the month
        last day in October
        last day in October 1996

last DoW in MMM [YYYY]
     The last DoW in the month
        last Tuesday in October
        last Tuesday in October 1996

last DoW in YYYY
     The last DoW in the year
        last Tuesday in 1997

        NOTE: "last DoW" doesn't work in
        English since the word "last"
        is used for both this expression
        and for "prev DoW", which gets
        parsed first. "last DoW" MAY
        work in other languages.

Nth DoW in MMM [YYYY]
     The Nth DoW in the month
        3rd Tuesday in October
        3rd Tuesday in October 1996

Nth DoW [YYYY]
     The Nth DoW in the year
        22nd Sunday
```

```
                22nd Sunday in 1996

        Nth day in MMM [YYYY]
             The Nth day of the month
                1st day of February
                1st day of February 2012

        DoW week
             British: same as "in 1 week on DoW"
                Monday week

        DoW week N [YYYY]
        Dow Nth week [YYYY]
             Sunday week 22
             Sunday 22nd week
                These refer to the day of week
                of the Nth week of the year.

        Nth
             12th
                This refers to the Nth day of the
                current month.
```

Note that the formats ‘‘Sunday week 22’’ and ‘‘22nd Sunday’’ give different behaviors. ‘‘Sunday week 22’’ returns the Sunday of the 22nd week of the year based on how week 1 is defined. ISO 8601 defines week one to contain Jan 4, so ‘‘Sunday week 1’’ might be the first or second Sunday of the current year, or the last Sunday of the previous year. ‘‘22nd Sunday’’ gives the actual 22nd time Sunday occurs in a given year, regardless of the definition of a week.

Special date strings

Most languages have strings which can be used to specify the date (relative to today). In English, these include the strings:

```
today
tomorrow
yesterday
```

There is also support for the British formats:

```
today week
tomorrow week
yesterday week
```

which refer to one week after today/tomorrow/yesterday respectively.

Other languages have similar strings.

Holidays

You can parse holiday names as dates (including timezones). For example:

```
Christmas
Christmas 2010
Christmas 2010 at noon
Christmas 2010 at noon PST
Saturday Christmas 2010 at noon
```

In all of the formats (except for ISO 8601 formats), the day of week (‘‘Friday’’) can be entered anywhere in the date and it will be checked for accuracy. In other words,

```
  "Tue Jul 16 1996 13:17:00"
```

will work but

```
  "Jul 16 1996 Wednesday 13:17:00"
```

will not (because Jul 16, 1996 is Tuesday, not Wednesday).

A NOTE ABOUT FOREIGN LANGUAGE DATES

Although Date::Manip has some support for parsing dates in foreign languages, it must be noted that the formats supported are largely based on English equivalents.

There are probably many different dates that are perfectly valid, and in common usage, in other languages which do not have an equivalent in the English language, and unfortunately, Date::Manip will probably not parse these.

You are free to send these to me, and I'll see if there is a way to add them in, but I do not guarantee anything. Without having a full-blown language parser (or at least the portion of the language that is devoted to calendar and time), most of these formats will simply not be supportable.

VALID TIME FORMATS

There are several categories of time formats supported by Date::Manip. These are strings which specify only the hour/minute/second fields.

ISO 8601 times

A time may be also be complete or truncated. Again, Date::Manip treats some formats as complete even though the specification calls them truncated.

A time may include the following fields:

```
   HH    2-digit representation of the hour
   MN    2-digit representation of the minutes
   SS    2-digit representation of the seconds
   H+    1+ digit representation of fractional hours
   M+    1+ digit representation of fractional minutes
   S+    1+ digit representation of fractional seconds
```

The following time formats are considered complete by Date::Manip. The time 12:30:15 will be expressed in the examples.

```
   Format      Notes   Examples

   HHMNSS      2       123015

   HH:MN:SS            12:30:15

   HHMNSS,S+           123015,5
   HH:MN:SS,S+         12:30:15,5
                       Fractional seconds are ignored

   HHMN,M+             1230,25
   HH:MN,M+            12:30,25
                       This is 12:30:00 + 0.25 minutes

   HH,H+               12,5
                       This is 12:00:00 + 0.5 hours, so
                       this is equivalent to 12:30:00

   -MNSS       1       -3015
   -MN:SS              -30:15
   --SS         1       --15

   -MNSS,S+     1       -3015,5
   -MN:SS,S+            -30:15,5

   -MN,M+       1       -30,25

   --SS,S+      1       --15,5

   HHMN         3       1230
   HH:MN                12:30
```

The following time formats are truncated:

```
   HH                   12

   -MN                  -30
```

Notes:

1 These formats are considered truncated in the standard, but since
  they do include (or imply, using the current time for defaults) all of
  the fields, and since they do not introduce any parsing complexities,
  the standard is relaxed, and they are treated as complete.

2 The HHMNSS format will not be correctly parsed since it is impossible
  to distinguish between it and YYMMDD. In order to parse an all-digit
  time, add the string ‘‘,0’’ to the end to force it to be interpreted
  as a time or include time zone information (either a zone name or
  abbreviation... an offset will not work in this case).

3 The HH:MN format will be treated as complete, even though it is
  incomplete due to missing the seconds. In real life, expressing
  a time in the HH:MN format is very common, and is regarded as complete,
  and might include time zone information.

ISO 8601 times may be followed by a time zone unless they are truncated. Truncated times may not include a timezone. Date::Manip relaxes the constraints placed on the time zone format and allows any of the methods used to specify the time zone including time zone name, abbreviation, or offset. The time zone may be separated from the time by a space, but it is not required.

Another constraint that is relaxed is that the fractional part may be specified using a period. In other words, the following are equivalent:

```
   12:30,25
   12:30.25
```

It should be noted (as it is in the specification) that using a negative time zone offset may cause confusion. In addition to visually confusing, it may not be parsed correctly. For example, the time:

```
   123005-0300
```

may not be parsed correctly. When using an offset time zone, you should always use the colon separators in the time:

```
   12:30:05-0300
```

Other time formats

A time may include any of the following fields:

```
H24   1− or 2−digit representation of the hour (0−23)
H12   1− or 2−digit representation of the hour (1−12)
MN    2−digit representation of the minutes
SS    2−digit representation of the seconds
H+    1+ digit representation of fractional hours
M+    1+ digit representation of fractional minutes
S+    1+ digit representation of fractional seconds
AM    A language specific AM/PM string
```

The following time formats are accepted:

```
Format              Examples

H24:MN:SS           17:30:15
H12:MN:SS AM        5:30:15 PM
H12:MN:SS

H24:MN:SS,S+        17:30:15,5
H12:MN:SS,S+ AM     5:30:15,5 PM
H12:MN:SS,S+        Fractional seconds are ignored

H24:MN,M+           17:30,25
H12:MN,M+ AM        5:30,25 PM
H12:MN,M+           This is 17:30:00 + 0.25 minutes

H24,H+              17,5
H12,H+ AM           5,5 PM
H12,H+              This is 17:00:00 + 0.5 hours, so
                    this is equivalent to 17:30:00

H24:MN              17:30
H12:MN AM           5:30 PM
H12:MN

H12 AM              5 PM
```

The fractional part may be specified using a comma or a period. Fractional seconds may also be separated using a colon. A language specific fractional separator may also be available for some languages.

In other words, the following are equivalent:

```
12:30:20,25
12:30:20.25
12:30:20:25
```

Some languages have alternate H:MN and MN:S separators. For example, one H:MN separator in French is 'h' (the MN:S separator is still a colon), so the following are equivalent:

```
12:30:00
12h30:00
```

Time zone information can be included immediately following the time. It can be separated by whitespace from the time, or it can be immediately adjacent.

Special time strings

Different languages may have some words which can be used to specify a certain time of day. In English, for example, the following words are equivalent to the time listed:

```
   noon        12:00:00
   midnight    00:00:00
```

So, the following are equivalent:

```
   Jan 2 2009 at noon
   Jan 2 2009 12:00:00
```

There were two possible ways to interpret midnight. One was at the start of the day (00:00:00) and the other was at the end of the day (24:00:00 which would actually mean at 00:00:00 of the following day). The first has been used to maintain backwards compatibility with Date::Manip 5.xx .

Other languages have similar strings.

In most languages, a word similar to ‘‘at’’ may precede the time (this does NOT apply to ISO 8601 time formats). This word (which must be separate from all other parts of the date with whitespace) is optional, and the following are equivalent:

```
   12:30
   at 12:30
```

The times ‘‘12:00 am’’, ‘‘12:00 pm’’, and ‘‘midnight’’ are not well defined. Date::Manip uses the following convention:

```
  midnight = 12:00am = 00:00:00
  noon     = 12:00pm = 12:00:00
```

and the day goes from 00:00:00 to 23:59:59. In other words, midnight is the beginning of a day rather than the end of one. The time 24:00:00 is also allowed (though it is automatically transformed to 00:00:00 of the following day). This gives the unusual result of parsing:

```
  Wed Feb 8 2006 24:00:00
```

which gives the date of:

```
  Thu Feb 9 2006 00:00:00
```

VALID COMBINED DATE AND TIME FORMATS

There are several categories of strings which specify both the date and time. These include the following:

ISO 8601 combined date and time

A combined ISO 8601 date and time is a string containing a complete ISO 8601 date and a complete or truncated ISO 8601 time. It may also include a timezone, provided a complete time is included.

Date::Manip relaxes the restrictions on how the two are combined. The time may be separated from the date by space, dash, or the letter T, or the two may be joined with nothing separating them.

When the time immediately follows the date, or when the two are separated by a dash, the resulting string MUST be unambiguous. Provided the date includes all of the dashes in it (i.e. YY-MM-DD instead of YYMMDD), it is rare that there is any ambiguity. If the date does not include dashes, the strings may be ambiguous, and in this case, separating the date and time with a space or the letter T is useful (and perhaps necessary) to correctly interpret the string.

The DoY formats should always be separated from the time by something. They are visually confusing if they are not separated from the time.

Time zone information can be included immediately following a complete time. It may not be included if no time is given, or if a truncated time is included. The time zone may be separated from the time with whitespace, or it can be immediately adjacent to it (since the ISO 8601 specification allows it in some cases).

Non-ISO 8601 combined date and time

A date from any of the non-ISO 8601 formats above may be combined with any of the non-ISO 8601 time formats above in any combination to form a valid combined date and time.

Deltas

Dates are often specified in terms of a delta from "now". For example, "in 2 days".

Most valid deltas can be used to specify a date, and the date is defined as that delta added to "now". Refer to the Date::Manip::Delta documentation for a list of valid delta formats.

If the delta itself does not include a time part, the time may be specified explicitly. For example:

```
   in 3 days at 12:00:00
   in 3 days at 12:00:00 PST
```

will take the delta part "in 3 days" and add it to the current time, then set the time to 12:00:00.

It is NOT allowed to include an explicit time if any time segment was included in the delta. For example, the following is invalid:

```
   in 3 days 2 hours at 12:00:00
```

One additional format that is supported is to include only week (or higher) components in the delta and to set the day of week. For example:

```
  Friday in 2 weeks
  in 2 weeks on Friday
  Friday 2 weeks ago
  2 weeks ago on Friday at 13:45
```

These first apply the delta (of weeks, months, and years) to the current time, and then set the day to the given day-of-week in that week.

Special date and time strings

Most language have strings which can be used to specify the full date and time (relative to the current date and time). In English, these include the string:

```
   now
```

They may also have a timezone attached:

```
   now PST
```

Additional combined date and time formats

The following formats are also supported:

```
   epoch SECS
      The number of seconds since the epoch
      (Jan 1, 1970 00:00:00 GMT). SECS may
      be negative to give time before the
      epoch.
```

or

```
   epoch SECS TIMEZONE
```

A couple of notes:

Commas may be included in all date formats arbitrarily (except for ISO 8601 formats where they may only be included when allowed by the specification).

The time/time zone is removed from the date before the date is parsed, so the time may appear before or after the date, or between any two parts of the date.

The time and the zone do not need to be adjacent, so the string:

```
   Jan 21 17:13:27 2010 −0400
```

will work. If the timezone is separate from the date, it MUST be separated from any other portion of the date by whitespace.

Certain words such as ‘‘on’’, ‘‘in’’, ‘‘at’’, ‘‘of’’, etc. which commonly appear in a date or time are ignored (except in ISO 8601 formats).

PRINTF DIRECTIVES

The following printf directives are replaced with information from the date.

```
   Year
       %y     year                        - 00 to 99
       %Y     year                        - 0001 to 9999

   Month, Week
       %m     month of year               - 01 to 12
       %f     month of year               - " 1" to "12"
       %b,%h  month abbreviation          - Jan to Dec
       %B     month name                  - January to December

   Day
       %j     day of the year             - 001 to 366
       %d     day of month                - 01 to 31
       %e     day of month                - " 1" to "31"
       %v     weekday abbreviation        - " S"," M"," T", ...
       %a     weekday abbreviation        - Sun to Sat
       %A     weekday name                - Sunday to Saturday
       %w     day of week                 - 1 to 7 (1=Monday)
       %E     day of month with
              suffix                      - 1st, 2nd, 3rd...

   Hour
       %H     hour                        - 00 to 23
       %k     hour                        - " 0" to "23"
       %i     hour                        - " 1" to "12"
       %I     hour                        - 01 to 12
       %p     AM or PM

   Minute, Second, Time zone
       %M     minute                      - 00 to 59
       %S     second                      - 00 to 59
       %Z     time zone abbreviation      - EDT
       %z     time zone as GMT offset     - +0100 (see Note 4)
       %N     time zone as GMT offset     - +01:00:00

   Epoch (see NOTE 3 below)
       %s     seconds from
              1/1/1970 GMT                - negative if before
       %o     seconds from 1/1/1970
              in the current time
              zone

   Date, Time
       %c     %a %b %e %H:%M:%S %Y        - Fri Apr 28 17:23:15 1995
       %C,%u  %a %b %e %H:%M:%S %Z %Y     - Fri Apr 28 17:25:57 EDT 1995
       %g     %a, %d %b %Y %H:%M:%S %Z    - Fri, 28 Apr 1995 17:23:15 EDT
       %D     %m/%d/%y                    - 04/28/95
       %x     %m/%d/%y or %d/%m/%y        - 04/28/95 or 28/04/95
                                            (Depends on DateFormat variable)
       %l     date in ls(1) format (see NOTE 1 below)
```

```
                 %b %e %H:%M            - Apr 28 17:23 (*)
                 %b %e  %Y              - Apr 28  1993 (*)
    %r     %I:%M:%S %p                  - 05:39:55 PM
    %R     %H:%M                        - 17:40
    %T,%X  %H:%M:%S                     - 17:40:58
    %V     %m%d%H%M%y                   - 0428174095
    %Q     %Y%m%d                       - 19961025
    %q     %Y%m%d%H%M%S                 - 19961025174058
    %P     %Y%m%d%H:%M:%S               - 1996102517:40:58
    %O     %Y-%m-%dT%H:%M:%S            - 1996-10-25T17:40:58
    %F     %A, %B %e, %Y                - Sunday, January  1, 1996
    %K     %Y-%j                        - 1997-045

Special Year/Week formats (see NOTE 2 below)
    %G     year, Monday as first
           day of week                  - 0001 to 9999
    %W     week of year, Monday
           as first day of week         - 01 to 53
    %L     year, Sunday as first
           day of week                  - 0001 to 9999
    %U     week of year, Sunday
           as first day of week         - 01 to 53
    %J     %G-W%W-%w                    - 1997-W02-2

Other formats
    %n     insert a newline character
    %t     insert a tab character
    %%     insert a `%' character
    %+     insert a `+' character

All other characters are currently unused, but may be used in the
future.  They currently insert the character following the %.

The following multi-character formats also exist:

Extended formats
    %<A=NUM>   These returns the NUMth value of the %A, %a, and %v formats
    %<a=NUM>   respectively.  In English, that would yield:
    %<v=NUM>      %<A=2>   => Tuesday
                  %<a=2>   => Tue
                  %<v=2>   => T
               NUM must be in the range 1-7.

    %<B=NUM>   These return the NUMth value of the %B and %b formats
    %<b=NUM>   respectively.  In English, that would yield:
                  %<B=2>   => February
                  %<b=2>   => Feb
               NUM must be in the range 1-12 (or 01-12).

    %<p=NUM>   These return the NUMth value of the %p format.  In
               English, that would yield:
                  %<p=1>   => AM
                  %<p=2>   => PM
               NUM must be in the range 1-2.
```

```
        %<E=NUM>   These return the NUMth value of the %E format.  In
                   English, that would yield:
                      %<E=1>   => 1st
                      %<E=53>  => 53rd
                   NUM must be in the range 1−53.
```

If a lone percent is the final character in a format, it is ignored.

The formats used in this routine were originally based on date.pl (version 3.2) by Terry McGonigal, as well as a couple taken from different versions of the Solaris **date** (1) command. Also, several have been added which are unique to Date::Manip.

NOTE 1:

The ls format (%l) applies to date within the past OR future 6 months! Any date that is before the date NOW − 6 months, or that is on or after the date NOW + 6 months will have the year printed out.

The later time must be on or after so that there is no ambiguity. If it is now 2000−06−06−12:00:00, then the date 1999−12−06−12:00:00 will be written as ‘‘Dec 6 12:00’’ but the date 2000−12−06−12:00:00 will be written as ‘‘Dec 6 2000’’.

NOTE 2:

The `%U`, `%W`, `%L`, `%G`, and `%J` formats are used to support the ISO−8601 format: YYYY-wWW-D. In this format, a date is written as a year, the week of the year, and the day of the week. Technically, the week may be considered to start on any day of the week, but Sunday and Monday are the both common choices, so both are supported.

The `%W` and `%G` formats return the week-of-year and the year treating weeks as starting on Monday.

The `%U` and `%L` formats return the week-of-year and the year treating weeks as starting on Sunday.

Most of the time, the `%L` and `%G` formats returns the same value as the `%Y` format, but there is a problem with days occurring in the first or last week of the year.

The ISO−8601 representation of Jan 1, 1993 written in the YYYY-wWW-D format is actually 1992−W53−5. In other words, Jan 1 is treated as being in the last week of the preceding year. Depending on the year, days in the first week of a year may belong to the previous year, and days in the final week of a year may belong to the next year. The week is assigned to the year which has most of the days. For example, if the week starts on Sunday, then the last week of 2003 is 2003−12−28 to 2004−01−03. This week is assigned to 2003 since 4 of the days in it are in 2003 and only 3 of them are in 2004. The first week of 2004 starts on 2004−01−04.

The `%U` and `%W` formats return a week-of-year number from 01 to 53. `%L` and `%G` return the corresponding year, and to get this type of information, you should always use the (%W,%G) combination or (%U,%L) combination. `%Y` should not be used as it will yield incorrect results.

`%J` returns the full ISO−8601 format (%G−W%W−%w).

NOTE 3:

The `%s` and `%o` formats return negative values if the date is before the start of the epoch. Other Unix utilities would return an error, or a zero, so if you are going to use Date::Manip in conjunction with these, be sure to check for a negative value.

NOTE 4:

The `%z` format returns the offset in the RFC 822 specified format +0500 . Most offsets are full hour amounts, so this is not a problem, but some offsets are irregular (+05:17:30). In this case, the string returned is +051730 which isn’t RFC 822 compliant, but since RFC 822 ignores this situation, I had to decide between returning an incorrect value, or breaking strict compliance, and I chose the second option.

KNOWN BUGS

None known.

BUGS AND QUESTIONS

Please refer to the Date::Manip::Problems documentation for information on submitting bug reports or questions to the author.

SEE ALSO

Date::Manip − main module documentation

LICENSE

AUTHOR

Sullivan Beck (sbeck@cpan.org)