NAME

Date::Manip::Base − Base methods for date manipulation

SYNOPSIS

```
use Date::Manip::Base;
$dmb = new Date::Manip::Base;
```

DESCRIPTION

The Date::Manip package of modules consists of several modules for doing high level date operations with full error checking and a lot of flexibility.

The high level operations, though intended to be used in most situations, have a lot of overhead associated with them. As such, a number of the most useful low level routines (which the high level routines use to do much of the real work) are included in this module and are available directly to users.

These low level routines are powerful enough that they can be used independent of the high level routines and perform useful (though much simpler) operations. They are also significantly faster than the high level routines.

These routines do little error checking on input. Invalid data will result in meaningless results. If you need error checking, you must call the higher level Date::Manip routines instead of these.

These routines also ignore all effects of time zones and daylight saving time. One way to think of these routines is working with times and dates in the GMT time zone.

BASE METHODS

This class inherits several base methods from the Date::Manip::Obj class. Please refer to the documentation for that class for a description of those methods.

err
new
new_config

Please refer to the Date::Manip::Obj documentation for these methods.

config

```
$dmb->config($var1,$val1,$var2,$val2,...);
```

This will set the value of any configuration variable. Please refer to the Date::Manip::Config manual for a list of all configuration variables and their description.

DATE METHODS

In all of the following method descriptions, the following variables are used:

$date

This is a list reference containing a full date and time:

```
[$y, $m, $d, $h, $mn, $s]
```

$ymd

A list reference containing only the date portion:

```
[$y, $m, $d]
```

$hms

A list reference containing only the time portion:

```
[$h, $mn, $s]
```

$delta

A list containing a full delta (an amount of time elapsed, or a duration):

```
[$dy, $dm, $dw, $dd, $dh, $dmn, $ds]
```

$time

A list reference containing the hour/minute/second portion of a delta:

```
[$dh, $dmn, $ds]
```

$offset

A list containing a time zone expressed as an offset:

```
[ $offh, $offm, $offs ]
```

Although this module does not make use of timezone information, a few of the functions perform operations on time zone offsets, primarily because these operations are needed in the higher level modules.

The elements ($y, $m, $d, $h, $mn, $s) are all numeric. In most of the routines described below, no error checking is done on the input. $y should be between 1 and 9999, $m between 1 and 12, $d between 1 and 31, $h should be between 0 and 23, $mn and $s between 0 and 59.

$hms can be between 00:00:00 and 24:00:00, but an $offset must be between −23:59:59 and +23:59:59.

Years are not translated to 4 digit years, so passing in a year of ‘‘04’’ will be equivalent to ‘‘0004’’, NOT ‘‘2004’’.

The elements ($dy, $dm, $dw, $dd, $dh, $dmn, $ds) are all numeric, but can be positive or negative. They represent an elapsed amount of time measured in years, months, weeks, etc.

Since no error checking is done, passing in ($y,$m,$d) = (2004,2,31) will NOT trigger an error, even though February does not have 31 days. Instead, some meaningless result will be returned.

calc_date_date
calc_date_days
calc_date_delta
calc_date_time
calc_time_time

These are all routines for doing simple date and time calculations. As mentioned above, they ignore all affects of time zones and daylight saving time.

The following methods are available:

```
$time = $dmb->calc_date_date($date1,$date2);
```

This take two dates and determine the amount of time between them.

```
$date = $dmb->calc_date_days($date,$n [,$subtract]);
$ymd  = $dmb->calc_date_days($ymd,$n [,$subtract]);
```

This returns a date $n days later (if $n>0) or earlier (if $n<0) than the date passed in. If $subtract is passed in, the sign of $n is reversed.

```
$date = $dmb->calc_date_delta($date,$delta [,$subtract]);
```

This take a date and add the given delta to it (or subtract the delta if $subtract is non-zero).

```
$date = $dmb->calc_date_time($date,$time [,$subtract]);
```

This take a date and add the given time to it (or subtract the time if $subtract is non-zero).

```
$time = $dmb->calc_time_time(@time1,@time2 [,$subtract]);
```

This take two times and add them together (or subtract the second from the first if $subtract is non-zero).

check
check_time

```
$valid = $dmb->check($date);
$valid = $dmb->check_time($hms);
```

This tests a list of values to see if they form a valid date or time ignoring all time zone affects. The date/time would be valid in GMT, but perhaps not in all time zones.

1 is returned if the the fields are valid, 0 otherwise.

`$hms` is in the range 00:00:00 to 24:00:00.

cmp

```
$flag = $dmb->cmp($date1,$date2);
```

Returns −1, 0, or 1 if date1 is before, the same as, or after date2.

day_of_week

```
$day = $dmb->day_of_week($date);
$day = $dmb->day_of_week($ymd);
```

Returns the day of the week (1 for Monday, 7 for Sunday).

day_of_year

```
$day = $dmb->day_of_year($ymd);
$day = $dmb->day_of_year($date);
```

In the first case, returns the day of the year (1 to 366) for `($y, $m, $d)`. In the second case, it returns a fractional day (1.0 <= `$day` < 367.0). For example, day 1.5 falls on Jan 1, at noon. The somewhat non-intuitive answer (1.5 instead of 0.5) is to make the two forms return numerically equivalent answers for times of 00:00:00 . You can look at the integer part of the number as being the day of the year, and the fractional part of the number as the fraction of the day that has passed at the given time.

The inverse operations can also be done:

```
$ymd   = $dmb->day_of_year($y,$day);
$date  = $dmb->day_of_year($y,$day);
```

If `$day` is an integer, the year, month, and day is returned. If `$day` is a floating point number, it returns the year, month, day, hour, minutes, and decimal seconds.

`$day` must be greater than or equal to 1 and less than 366 on non-leap years or 367 on leap years.

days_in_month

```
$days = $dmb->days_in_month($y,$m);
```

Returns the number of days in the month.

```
@days = $dmb->days_in_month($y,0);
```

Returns a list of 12 elements with the days in each month of the year.

days_in_year

```
$days = $dmb->days_in_year($y);
```

Returns the number of days in the year (365 or 366)

days_since_1BC

```
$days = $dmb->days_since_1BC($date);
$days = $dmb->days_since_1BC($ymd);
```

Returns the number of days since Dec 31, 1BC. Since the calendar has changed a number of times, the number returned is based on the current calendar projected backwards in time, and in no way reflects a true number of days since then. As such, the result is largely meaningless, except when called twice as a means of determining the number of days separating two dates.

The inverse operation is also available:

```
$ymd = $dmb->days_since_1BC($days);
```

Returns the date `$days` since Dec 31, 1BC. So day 1 is Jan 1, 0001.

leapyear

```
$flag = $dmb->leapyear($y);
```

Returns 1 if the argument is a leap year.

nth_day_of_week

```
$ymd = $dmb->nth_day_of_week($y,$n,$dow);
```

Returns the `$n`th occurrence of `$dow` (1 for Monday, 7 for Sunday) in the year. `$n` must be between 1 and 53 or −1 through −53.

```
$ymd = $dmb->nth_day_of_week($y,$n,$dow,$m);
```

Returns the `$n`th occurrence of `$dow` in the given month. `$n` must be between 1 and 5 or it can be −1 through −5.

In all cases, nothing is returned if `$n` is beyond the last actual result (i.e. the 5th Sunday in a month with only four Sundays).

secs_since_1970

```
$secs = $dmb->secs_since_1970($date);
```

Returns the number of seconds since Jan 1, 1970 00:00:00 (negative if date is earlier).

```
$date = $dmb->secs_since_1970($secs);
```

Translates number of seconds into a date.

split
join

The split and join functions are used to take a string containing a common type of time data and split it into a list of fields. The join function takes the list and forms it into a string.

The general format for these is:

```
$obj    = $dmb->split($type,$string,\%opts);
$string = $dmb->join($type,$obj,\%opts);
```

An older format is also supported:

```
$obj    = $dmb->split($type,$string,[$no_normalize]);
$string = $dmb->join($type,$obj,[$no_normalize]);
```

but this is deprecated and will be removed in Date::Manip 7.00. These are equivalent to:

```
$obj    = $dmb->split($type,$string,{ 'nonorm' => $no_normalize });
$string = $dmb->join($type,$obj,{ 'nonorm' => $no_normalize });
```

The value of `$type` determines what type of join/split operation occurs.

Rudimentary error checking is performed with both of these functions and undef is returned in the case of any error. No error checking is done on the specific values.

`$type` = 'date'

```
$date = $dmb->split("date",$string);
$string = $dmb->join("date",$date);
```

This splits a string containing a date or creates one from a list reference. The string split must be of one of the forms:

```
YYYYMMDDHH:MN:SS
YYYYMMDDHHMNSS
YYYY-MM-DD-HH:MN:SS
```

The string formed by join is one of the above, depending on the value of the Printable config variable. The default format is YYYYMMDDHH:MN:SS, but if Printable is set to 1,

YYYYMMDDHHMNSS is produced, and if Printable is set to 2, the YYYY−MM−DD−HH:MN:SS form is produced.

$type = 'hms'

```
$hms = $dmb->split("hms",$string);
$string = $dmb->join("hms",$hms);
```

This works with the hours, minutes, and seconds portion of a date.

When splitting a string, the string can be of any of the forms:

```
H
H:MN
H:MN:SS
HH
HHMN
HHMNSS
```

Here, H is a 1 or 2 digit representation of the hours but HH (and all other fields) are two digit representations.

The string formed by the join function will always be of the form HH:MN:SS.

The time must be between 00:00:00 and 24:00:00.

$type = 'offset'

```
$offset = $dmb->split("offset",$string);
$string = $dmb->join("offset",$offset);
```

An offset string should have a sign (though it is optional if it is positive) and is any of the forms:

```
+H
+H:MN
+H:MN:SS
+HH
+HHMN
+HHMNSS
```

Here, H is a 1 or 2 digit representation of the hours. All other fields are two digit representations.

The string formed by the join function will always be of the form +HH:MN:SS.

The offset must be between −23:59:59 and +23:59:59 .

$type = 'time'

```
$time = $dmb->split("time",$string,\%opts]);
$string = $dmb->join("time",$time,\%opts);
```

The only option supported is:

```
'nonorm'   0/1
```

This works with an amount of time in hours, minutes, and seconds. The string is of the format:

```
+H:MN:S
```

where all signs are optional. The returned value (whether a list reference from the split function, or a string from the join function) will have all fields normalized unless `no_norm` is true.

$type = 'delta'

```
$delta = $dmb->split("delta",$string,\%opts);
$string = $dmb->join("delta",$delta,\%opts);
```

Options recognized are:

```
mode     : standard/business
nonorm   : 0/1
type     : exact/semi/approx/estimated
```

A second format is also supported, but is deprecated and will be removed in Date::Manip 7.0.

```
$delta = $dmb->split("business",$string,\%opts);
$string = $dmb->join("business",$delta,\%opts);
```

These are equivalent to using 'delta' with an option of `'mode' =` 'business'>.

These split a string containing a delta, or create a string containing one using the rules described in the Date::Manip::Delta documentation.

The string that can be split is of the form:

```
Y:M:W:D:H:MN:S
```

Any field may have a sign, but they are optional.

Fields may be omitted entirely. For example:

```
D:H:MN:S
D:::S
```

are both valid.

The string or list output is normalized unless the **nonorm** option is passed in.

The type of the delta (which determines how it will be normalized) will be automatically determined if not specified. The type will default to the value given in the table below based on the FIRST condition that is true.

```
default_type  condition

estimated     any field is a non-integer
approx        any of the approximate fields are non-zero
semi          any of the semi-exact fields are non-zero
exact         only the exact fields are non-zero
```

week1_day1

```
$ymd = $dmb->week1_day1($y);
```

This returns the date of the 1st day of the 1st week in the given year. Note that this uses the ISO 8601 definition of week, so the year returned may be the year before the one passed in.

This uses the FirstDay and Jan1Week1 config variables to evaluate the results.

weeks_in_year

```
$w = $dmb->weeks_in_year($y);
```

This returns the number of ISO 8601 weeks in the year. It will always be 52 or 53.

week_of_year

```
($y,$w) = $dmb->week_of_year($date);
($y,$w) = $dmb->week_of_year($ymd);
```

This returns the week number (1-53) of the given date and the year that it falls in. Since the ISO 8601 definition of a week is used, the year returned is not necessarily the one passed in (it may differ for the

first or last week of the year).

The inverse operation is also available:

```
$ymd = $dmb->week_of_year($y,$w);
```

which returns the first day of the given week.

This uses the FirstDay and Jan1Week1 config variables to evaluate the results.

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)