NAME

Date::Manip::Recur − methods for working with recurring events

SYNOPSIS

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

DESCRIPTION

This module contains functions useful in parsing and manipulating recurrences. A recurrence is a notation for specifying when a recurring event occurs. For example, if an event occurs every other Friday or every 4 hours, this can be defined as a recurrence. A fully specified recurrence consists of the following pieces of information:

Frequency

The most basic piece of information is the frequency. For relatively simple recurring events, the frequency defines when those events occur. For more complicated recurring events, the frequency tells approximately when the events occur (but to get the actual events, certain modifiers must be applied as described below).

Examples of recurring events include:

```
the first of every month
every other day
the 4th Thursday of each month at 2:00 PM
every 2 hours and 30 minutes
```

All of these can be expressed as a frequency.

NOTE: unlike date parsing, support for frequencies written out in English (or whatever language you are working in) is extremely limited. For example, the string ‘‘the first of every month’’ will NOT be parsed as a valid frequency. A limited number of frequencies can be expressed in a written out form (see the <L/‘‘OTHER FREQUENCY FORMATS’’ section below), but most must be expressed in the format described below in <L/‘‘FREQUENCY NOTATION’’>. In this document however, the written out form will often be used for the sake of clarity.

Since a frequency typically refers to events that could happen an infinite number of times, you usually have to specify a date range to get the actual dates. Some frequencies also require a base date (i.e. information about when one such even actually occurred) since the frequency is otherwise ambiguous. For example, the frequency ’every other day’ does not include enough information to specify the dates that the event happened on, so you have to explicitly define one of them. Then all others can be derived.

Modifier

Complex recurring events may require the use of modifiers in order to get them correct.

For example, in America, many places treat both Thanksgiving and the day after as holidays. Thanksgiving is easy to define since it is defined as:

```
4th Thursday of every November
```

In the frequency notation (described below), this would be written as:

```
1*11:4:5:0:0:0
```

The day after Thanksgiving is NOT possible to define in the same way. Depending on the year, the day after the 4th Thursday may be the 4th or 5th Friday.

The only way to accurately define the day after Thanksgiving is to specify a frequency and a modifier:

```
4th Thursday of every November
+1 day
```

In frequency notation, this can be expressed as:

```
1*11:4:5:0:0:0*FD1
```

The syntax for the various modifiers is described below in the ‘‘MODIFIERS’’ section.

Base date

Many recurrences have a base date which is a date on which a recurring event is based.

The base date is not necessarily a date where the recurring event occurs. Instead, it may be modified (with modifiers, or with values specified in the recurrence) to actually produce a recurring event.

For example, if the frequency is

```
every other Friday at noon
```

the base date will be a Friday and the recurring event will happen on that Friday, Friday two weeks later, Friday four weeks later, etc. In all cases, the dates will be modified to be at noon.

If the frequency has a modifier, such as:

```
every other Friday
+ 1 day
```

(and yes, this trivial example could be expressed as the frequency ’every other Saturday’ with no modifiers), then the base date is still on a Friday, but the actual recurring event is determined by applying modifiers and occurs on Saturday.

Recurring events are assigned a number with the event that is referred to by the base date being the 0th occurrence, the first one after that as the 1st occurrence, etc. Recurring events can also occur before the base date with the last time the recurring event occurred before the base date is the −1th occurrence.

So, if the frequency is

```
the first of every month
```

and the base date is ’Mar 1, 2000’, then the 5 recurring events around it are:

```
N    Date

−2   Jan 1 2000
−1   Feb 1 2000
 0   Mar 1 2000
+1   Apr 1 2000
+2   May 1 2000
```

In some cases, the Nth date may not be defined. For example, if the frequency is:

```
the 31st of every month
```

and the base date is Mar 31, 2000, the 5 recurring events around it are:

```
N   Date

−2  Jan 31 2000
−1  undefined
 0  Mar 31 2000
 1  undefined
 2  May 31 2000
```

As mentioned above, the base date is used to determine one of the occurrences of the recurring event... but it may not actually be on of those events.

As an example, for the recurring event:

```
every other Friday
```

a base date could be on a Friday, but it would also be possible to have a base date on some other day of the week, and it could unambiguously refer simply to a week, and the recurring event would occur on Friday of that week.

In most cases, it won't be necessary to treat base dates with that level of complexity, but with complicated recurring events, it may be necessary. More information on how Date::Manip determines a recurring event from a base date is given below in the section "BASE DATES".

Range

A date range is simply a starting and an ending date. When a range is used (primarily in the dates method as described below), only recurring events (with all modifiers applied) which happened on or after the start date and on or before the end date are used.

For example, if the frequency was

```
the first of every month
```

and the start/end dates were Jan 1 2000 and May 31 2000, the list of dates referred to would be:

```
Jan 1 2000
Feb 1 2000
Mar 1 2000
Apr 1 2000
May 1 2000
```

If no base date is specified, but a date range is specified, the start date is used as the specified base date.

It should be noted that if both the range and base date are specified, the range is not used to determine a base date. Also, the first time the recurring event occurs in this range may NOT be the 0th occurrence with respect to the base date, and that is allowed.

NOTE: both dates in the range and the base date must all be in the same time zone, and use the same Date::Manip::Base object.

An alternate definition of the range may also be used to specify that the recurring events based only on the interval and BEFORE any modifiers are applied fall in the range.

This definition is described in more detail below.

FREQUENCY NOTATION

The syntax for specifying a frequency requires some explanation. It is very concise, but contains the flexibility to express every single type of recurring event I could think of.

The syntax of the frequency description is a colon separated list of the format Y:M:W:D:H:MN:S (which stand for year, month, week, etc.). One (and only one) of the colons may optionally be replaced by an asterisk, or an asterisk may be prepended to the string. For example, the following are all valid frequency descriptions:

```
  1:2:3:4:5:6:7
  1:2*3:4:5:6:7
 *1:2:3:4:5:6:7
```

But the following are NOT valid because they contain more than one asterisk:

```
  1:2*3:4:5*6:7
 *1:2:3:4:5:6*7
```

When an asterisk is included, the portion to the left of it is called the interval, and refers to an approximate time interval between recurring events. For example, if the interval of the frequency is:

```
  1:2*
```

it means that the recurring event occurs approximately every 1 year and 2 months. The interval is approximate because elements to the right of the asterisk, as well as any modifiers included in the recurrence, will affect when the events actually occur.

If no asterisks are included, then the entire recurrence is an interval. For example,

```
0:0:0:1:12:0:0
```

refers to an event that occurs every 1 day, 12 hours.

The format of the interval is very simple. It is colon separated digits only. No other characters are allowed.

The portion of the frequency that occur after an asterisk is called the recurrence time (or rtime), and refers to a specific value (or values) for that type of time element (i.e. exactly as it would appear on a calendar or a clock). For example, if the frequency ends with the rtime:

```
*12:0:0
```

then the recurring event occurs at 12:00:00 (noon).

For example:

```
0:0:0:2*12:30:0      every 2 days at 12:30 (each day)
```

Elements in the rtime can be listed as single values, ranges (2 numbers separated by a dash ‘‘−’’), or a comma separated list of values or ranges. In some cases, negative values are appropriate for the week or day values. −1 stands for the last possible value, −2 for the second to the last, etc.

If multiple values are included in more than one field in the rtime, every possible combination will be used. For example, if the frequency ends with the rtime:

```
*12−13:0,30:0
```

the event will occur at 12:00, 12:30, 13:00, and 13:30.

Some examples are:

```
0:0:0:1*2,4,6:0:0    every day at at 02:00, 04:00, and 06:00
0:0:0:2*12−13:0,30:0 every other day at 12:00, 12:30, 13:00,
                     and 13:30
0:1:0*−1:0:0:0       the last day of every month
*1990−1995:12:0:1:0:0:0
                     Dec 1 in 1990 through 1995
```

There is no way to express the following with a single recurrence:

```
every day at 12:30 and 1:00
```

You have to use two recurrences to do this.

You can include negative numbers in ranges. For example, including the range −2−−1 means to go from the 2nd to the last to the last occurrence. Negative values are only supported in the week and day fields, and only in some cases.

You can even use a range like 2−−2 (which means to go from the 2nd to the 2nd to the last occurrence). However, this is STRONGLY discouraged since this leads to a date which produces a variable number of events. As a result, the only way to determine the Nth date is to calculate every date starting at the base date. If you know that every date produces exactly 4 recurring events, you can calculate the Nth date without needing to determine every intermediate date.

When specifying a range, the first value must be less than the second or else nothing will be returned.

When both the week and day elements are non-zero and the day is right of the asterisk, the day refers to the day of week. The following examples illustrate these type of frequencies:

```
0:1*4:2:0:0:0        4th Tuesday (day 2) of every month
0:1*-1:2:0:0:0       last Tuesday of every month
0:0:3*2:0:0:0        every 3rd Tuesday (every 3 weeks
                     on 2nd day of week)
1:0*12:2:0:0:0       the 12th Tuesday of each year
```

NOTE: The day of week refers to the numeric value of each day as specified by ISO 8601. In other words, day 1 is ALWAYS Monday, day 7 is ALWAYS Sunday, etc., regardless of what day of the week the week is defined to begin on (using the FirstDay config variable). So when the day field refers to the day of week, it's value (or values if a range or comma separated list are used) must be 1−7.

When the week element is zero and the month element is non-zero and the day element is right of the asterisk, the day value is the day of the month (it can be from 1 to 31 or −1 to −31 counting from the end of the month).

```
3*1:0:2:12:0:0       every 3 years on Jan 2 at noon
0:1*0:2:12,14:0:0    2nd of every month at 12:00 and 14:00
0:1:0*-2:0:0:0       2nd to last day of every month
```

NOTE: If the day given refers to the 29th, 30th, or 31st, in a month that does not have that number of days, it is ignored. For example, if you ask for the 31st of every month, it will return dates in Jan, Mar, May, Jul, etc. Months with fewer than 31 days will be ignored.

If both the month and week elements are zero, and the year element is non-zero, the day value is the day of the year (1 to 365 or 366 — or the negative numbers to count backwards from the end of the year).

```
1:0:0*45:0:0:0       45th day of every year
```

Specifying a day that doesn't occur in that year silently ignores that year. The only result of this is that specifying +366 or −366 will ignore all years except leap years.

If the week element is non-zero and to the right of the asterisk, and the day element is zero, the frequency refers to the first day of the given week of the month or week of the year:

```
0:1*2:0:0:0:0        the first day of the 2nd week of
                     every month
1:0*2:0:0:0:0        the first day of the 2nd week of
                     every year
```

Although the meaning of almost every recurrence can be deduced by the above rules, a set of tables describing every possible combination of Y/M/W/D meanings, and giving an example of each is included below in the section ''LIST OF Y/M/W/D FREQUENCY DEFINITIONS''. It also explains a small number of special cases.

NOTE: If all fields left of the asterisk are zero, the last one is implied to be 1. In other words, the following are equivalent:

```
0:0:0*x:x:x:x
0:0:1*x:x:x:x
```

and can be thought of as every possible occurrence of the rtime.

NOTE: When applying a frequency to get a list of dates on which a recurring event occurs, a delta is created from the frequency which is applied to get dates referred to by the interval. These are then operated on by the rtime and by modifiers to actually get the recurring events. The deltas will always be exact or approximate. There is no support for business mode recurrences. However, with the careful use of modifiers (discussed below), most recurring business events can be determined too.

BASE DATES

A recurrence of the form *Y:M:W:D:H:MN:S (which is technically speaking not a recurring event... it is just a date or dates specified using the frequency syntax) uses the first date which matches the frequency as the base date. Any base date specified will be completely ignored. A date range may be specified to work with a subset of the dates.

All other recurrences use a specified base date in order to determine when the 0th occurrence of a recurring event happens. As mentioned above, the specified base date may be determined from the start date, or specified explicitly.

The specified base date is used to provide the bare minimum information. For example, the recurrence:

```
0:0:3*4:0:0:0        every 3 weeks on Thursday
```

requires a base date to determine the week, but nothing else. Using the standard definition (Monday-Sunday) for a week, and given that one week in August 2009 is Aug 10 to Aug 16, any date in the range Aug 10 to Aug 16 will give the same results. The definition of the week defaults to Monday-Sunday, but may be modified using the FirstDay config variable.

Likewise, the recurrence:

```
1:3*0:4:0:0:0        every 1 year, 3 months on the 4th
                     day of the month
```

would only use the year and month of the base date, so all dates in a given month would give the same set of recurring dates.

It should also be noted that a date may actually produce multiple recurring events. For example, the recurrence:

```
0:0:2*4:12,14:0:0    every 2 weeks on Thursday at 12:00
                     and 14:00
```

produces 2 events for every date. So in this case, the base date produces the 0th and 1st event, the base date + an offset produces the 2nd and 3rd events, etc.

It must be noted that the base date refers ONLY to the interval part of the recurrence. The rtime and modifiers are NOT used in determining the base date.

INTERVAL

The interval of a frequency (everything left of the asterisk) will be used to generate a list of dates (called interval dates). When rtime values and modifiers are applied to an interval date, it produces the actual recurring events.

As already noted, if the rtime values include multiple values for any field, more than one event are produced by a single interval date.

It is important to understand is how the interval dates are calculated. The interval is trivially turned into a delta. For example, with the frequency 0:0:2*4:12:0:0, the interval is 0:0:2 which produces the delta 0:0:2:0:0:0:0.

In order to get the Nth interval date, the delta is multiplied by N and added to the base date. In other words:

```
D(0) = Jan 31
D(1) = Jan 31 + 1 month = Feb 28
D(2) = Jan 31 + 2 month = Mar 31
```

DATE RANGE

The start and end dates form the range in which recurring events can fall into.

Every recurring date will fall in the limit:

```
start <= date <= end
```

When a recurrence is created, it may include a default range, and this is handled by the RecurRange config variable.

By default, the date range applies to the final dates once all modifiers have been applied.

This behavior can be changed by applying the range to the unmodified dates.

An example of how this applies might be in defining New Year's Day (observed). The most useful definition of this would be:

```
1*1:0:1:0:0:0*DWD
```

which means Jan 1 modified to the nearest working day.

But if you wanted to find New Year's for 2005 using this definition by passing in a start date of 2005−01−01−00:00:00 and an end date of 2005−12−31−23:59:59, you won't find anything because New Year's day will actually be observed on 2004−12−31 (since Jan 1 is a Saturday).

To get around this, you can pass in a non-zero parameter with the recurrence which means that this range will be applied to the unmodified dates.

In effect, this discards the modifier (DWD), gets the dates that fall in the range, and for all that fall in the range, the modifiers are applied.

So:

```
1*1:0:1:0:0:0*DWD**2005−01−01−00:00:00*2005−12−31−23:59:59
```

will return no dates, but:

```
1*1:0:1:0:0:0*DWD**2005−01−01−00:00:00*2005−12−31−23:59:59*1
```

will return:

```
2004−12−31−00:00:00
```

OTHER FREQUENCY FORMATS

There are a small handful of English strings (or the equivalent in other languages) which can be parsed in place of a numerical frequency. These include:

```
every Tuesday in June [1997]
2nd Tuesday in June [1997]
last Tuesday in June [1997]

every Tuesday of every month [in 1997]
2nd Tuesday of every month [in 1997]
last Tuesday of every month [in 1997]

every day of every month [in 1997]
2nd day of every month [in 1997]
last day of every month [in 1997]

every day [in 1997]
every 2nd day [in 1977]
every 2 days [in 1977]
```

Each of these set the frequency. If the year is include in the string, it also sets the dates in the range to be the first and last day of the year.

In each of these, the numerical part (i.e. 2nd in all of the examples above) can be any number from 1 to 31. To make a frequency with a larger number than that, you have to use the standard format discussed above.

Due to the complexity of writing out (and parsing) frequencies written out, I do not intend to add additional frequency formats, and the use of these is discouraged. The frequency format described above is preferred.

MODIFIERS

Any number of modifiers may be added to a frequency to get the actual date of a recurring event. Modifiers are case sensitive.

Modifiers to set the day-of-week

The following modifiers can be used to adjust a date to a specific day of the week.

```
PDn   Means the previous day n not counting today
PTn   Means the previous day n counting today
NDn   Means the next day n not counting today
NTn   Means the next day n counting today
WDn   Day n (1-7) of the current week
```

In each of these, 'n' is 1-7 (1 being Sunday, 7 being Saturday).

For example, PD2/ND2 returns the previous/next Tuesday. If the date that this is applied to is Tuesday, it modifies it to one week in the past/future.

PT2/NT2 are similar, but will leave the date unmodified if it is a Tuesday.

Modifiers to move forward/backward a number of days

These modifiers can be used to add/subtract n days to a date.

```
FDn   Means step forward n days.
BDn   Means step backward n days.
```

Modifiers to force events to be on business days

Modifiers can also be used to force recurring events to occur on business days. These modifiers include:

```
FWn   Means step forward n workdays.
BWn   Means step backward n workdays.

CWD   The closest work day (using the TomorrowFirst
      config variable).
CWN   The closest work day (looking forward first).
CWP   The closest work day (looking backward first).

NWD   The next work day counting today
PWD   The previous work day counting today
DWD   The closest work day (using the TomorrowFirst config
      variable) counting today

IBD   This discards the date if it is not a business day.
NBD   This discards the date if it IS a business day.

IWn   This discards the date if it is not the n'th day
      of the week (n=1-7, 1 is Monday)
NWn   This discards the date if it IS the n'th day of the week
```

The CWD, CWN, and CWP modifiers will always change the date to the closest working day NOT counting the current date.

The NWD, PWD, and DWD modifiers always change the date to the closest working day unless the current date is a work day. In that case, it is left unmodified.

CWD, CWN, and CWP will usually return the same value, but if you are starting at the middle day of a 3-day weekend (for example), it will return either the first work day of the following week, or the last work day of the previous week depending on whether it looks forward or backward first.

All business day modifiers ignore the time, so if a date is initially calculated at Saturday at noon, and the FW1 is applied, the date is initially moved to the following Monday (assuming it is a work day) and the FW1 moves it to Tuesday. The final result will be Tuesday at noon.

The IBD, NBD, IWn, and NWn modifiers eliminate dates from the list immediately. In other words, if a recurrence has three modifiers:

```
FD1,IBD,FD1
```

then as a date is being tested, first the FD1 modifier is applied. Then, it is tested to see if it is a business day. If it is, the second FD1 modifier will be applied. Otherwise, the date will not be included in the list of recurring events.

Special modifiers

The following modifiers do things that cannot be expressed using any other combination of frequency and modifiers:

```
EASTER   Set the date to Easter for this year.
```

DETERMINING DATES

In order to get a list of dates referred to by the recurrence, the following steps are taken.

The recurrence is tested for errors

The recurrence must be completely specified with a base date (either supplied explicitly, or derived from a start date) and date range when necessary. All dates must be valid.

The actual base date is determined

Using information from the interval and the specified base date, the actual base date is determined.

The Nth date is calculated

By applying the delta that corresponds to the interval, and then applying rtime and modifier information, the Nth date is determined.

This is repeated until all desired dates have been obtained.

The nth method described below has more details.

The range is tested

Any date that fall outside the range is discarded.

NOTE: when the recurrence contains no interval, it is not necessary to specify the range, and if it is not specified, all of the dates are used. The range MAY be specified to return only a subset of the dates if desired.

LIST OF Y/M/W/D FREQUENCY DEFINITIONS

Because the week and day values may have multiple meanings depending on where the asterisk is, and which of the fields have non-zero values, a list of every possible combination is included here (though most can be determined using the rules above).

When the asterisk occurs before the day element, and the day element is non-zero, the day element can take on multiple meanings depending on where the asterisk occurs, and which leading elements (year, month, week) have non-zero values. It can refer to the day of the week, day of the month, or day of the year.

When the asterisk occurs before the week element, the week element of the frequency can also take on multiple meanings as well. When the month field and day fields are zero, it refers to the week of the year. Since the week of the year is well defined in the ISO 8601 spec, there is no ambiguity.

When the month field is zero, but the day field is not, the week field refers to the nth occurrence of the day of week referred to by the day field in the year.

When the month field is non-zero, the week field refers to the nth occurrence of the day of week in the month.

In the tables below only the first 4 elements of the frequency are shown. The actual frequency will include the hour, minute, and second elements in addition to the ones shown.

When all elements left of the asterisk are 0, the interval is such that it occurs the maximum times possible (without changing the type of elements to the right of the asterisk). Another way of looking at it is that the last 0 element of the interval is changed to 1. So, the interval:

```
0:0*3:0
```

is equivalent to

```
0:1*3:0
```

When the year field is zero, and is right of the asterisk, it means the current year.

All elements left of the asterisk

When all of the month, week, and day elements are left of the asterisk, the simple definitions of the frequency are used:

```
frequency     meaning

1:2:3:4       every 1 year, 2 months, 3 weeks,
              4 days
```

Any, or all of the fields can be zero.

Non-zero day, non-zero week

When both the day and week elements are non-zero, the day element always refers to the day of week. Values must be in the range (1 to 7) and no negative values are allowed.

The following tables shows all possible variations of the frequency where this can happen (where day 4 = Thursday).

When the week is left of the asterisk, the interval is used to get the weeks on the calendar containing a recurring date, and the day is used to set the day of the week. The following are possible:

```
frequency     meaning

1:2:3*4       every 1 year, 2 months, 3 weeks
              on Thur

1:0:3*4       every 1 year, 3 weeks on Thur

0:2:3*4       every 2 months, 3 weeks on Thur

0:0:3*4       every 3 weeks on Thur
```

When the week is right of the asterisk, and a non-zero month is left of the asterisk, the recurrence refers to a specific occurrence of a day-of-week during a month. The following are possible:

```
frequency     meaning

1:2*3:4       every 1 year, 2 months on the
              3rd Thursday of the month

0:2*3:4       every 2 months on the 3rd Thur
              of the month
```

When the week and month are both non-zero and right of the asterisk, the recurrence refers to an occurrence of day-of-week during the given month. Possibilities are:

```
frequency     meaning

1*2:3:4       every 1 year in February on
              the 3rd Thur

0*2:3:4       same as 1*2:3:4

*1:2:3:4      in Feb 0001 on the 3rd Thur
              of the month

*0:2:3:4      on the 3rd Thur of Feb in the
```

```
                current year
```

When the week is right of the asterisk, and the month is zero, the recurrence refers to an occurrence of the day-of-week during the year. The following are possible:

```
  frequency     meaning

  1:0*3:4       every 1 year on the 3rd Thursday
  1*0:3:4       of the year

 *1:0:3:4       in 0001 on the 3rd Thur of
                the year

  0*0:3:4       same as 1*0:3:4

 *0:0:3:4       on the 3rd Thur of the current
                year
```

There is one special case:

```
  frequency     meaning

  0:0*3:4       same as 0:1*3:4 (every month on
                the 3rd Thur of the month)
```

Non-zero day, non-zero month

When a non-zero day element occurs to the right of the asterisk and the week element is zero, but the month element is non-zero, the day elements always refers to a the day of month in the range (1 to 31) or (−1 to −31).

The following table shows all possible variations of the frequency where this can happen:

```
  frequency     meaning

  1:2:0*4       every 1 year, 2 months on the
  1:2*0:4       4th day of the month

  1*2:0:4       every year on Feb 4th

 *1:2:0:4       Feb 4th, 0001

  0:2:0*4       every 2 months on the 4th day
  0:2*0:4       of the month

  0*2:0:4       same as 1*2:0:4

 *0:2:0:4       Feb 4th of the current year
```

Zero day, non-zero week

When a day is zero, and the week is non-zero, the recurrence refers to a specific occurrence of the first day of the week (as given by the FirstDay variable).

The frequency can refer to an occurrence of FirstDay in a specific week (if the week is left of the asterisk):

```
  frequency     meaning

  1:2:3*0       every 1 year, 2 months, 3 weeks on
                FirstDay
```

```
  1:0:3*0       every 1 year, 3 weeks on FirstDay

  0:2:3*0       every 2 months, 3 weeks on FirstDay

  0:0:3*0       every 3 weeks on FirstDay
```

or to a week in the year (if the week is right of the asterisk, and the month is zero):

```
  frequency     meaning

  1:0*3:0       every 1 year on the first day of the
  1*0:3:0       3rd week of the year

 *1:0:3:0       the first day of the 3rd week of 0001
```

or to an occurrence of FirstDay in a month (if the week is right of the asterisk and month is non-zero):

```
  frequency     meaning

  1:2*3:0       every 1 year, 2 months on the 3rd
                occurrence of FirstDay

  0:2*3:0       every 2 months on the 3rd occurrence
                of FirstDay

  1*2:3:0       every year on the 3rd occurrence
                of FirstDay in Feb

  0*2:3:0       same as 1*2:3:0

 *1:2:3:0       the 3rd occurrence of FirstDay
                Feb 0001

 *0:2:3:0       the 3rd occurrence of FirstDay
                in Feb of the current year
```

NOTE: in the last group, a slightly more intuitive definition of these would have been to say that the week field refers to the week of the month, but given the ISO 8601 manner of defining when weeks start, this definition would have virtually no practical application. So the definition of the week field referring to the Nth occurrence of FirstDay in a month was used instead.

There are a few special cases here:

```
  frequency     meaning

  0:0*3:0       same as 0:1*3:0   (every month on the 3rd
                occurrence of the first day of week)

  0*0:3:0       same as 1*0:3:0

 *0:0:3:0       the first day of the 3rd week of the
                current year
```

Non-zero day

When a non-zero day element occurs and both the month and week elements are zero, the day elements always refers to a the day of year (1 to 366 or −1 to −366 to count from the end).

The following table shows all possible variations of the frequency where this can happen:

```
 frequency     meaning

 1:0:0*4       every year on the 4th day of
 1:0*0:4       the year
 1*0:0:4

*1:0:0:4       the 4th day of 0001
```

Other non-zero day variations have multiple meanings for the day element:

```
 frequency     meaning

 0:0:0*4       same as 0:0:1*4  (every week on Thur)

 0:0*0:4       same as 0:1*0:4  (every month on the 4th)

 0*0:0:4       same as 1*0:0:4

*0:0:0:4       the 4th day of the current year
```

All other variations

The remaining variations have zero values for both week and day. They are:

```
 frequency     meaning

 1:2:0*0       every 1 year, 2 months on the first
 1:2*0:0       day of the month

 1*2:0:0       every year on Feb 1

*1:2:0:0       Feb 1, 0001

 1:0:0*0       every 1 year on Jan 1
 1:0*0:0
 1*0:0:0

*1:0:0:0       Jan 1, 0001

 0:2:0*0       every 2 months on the first day of
 0:2*0:0       the month

 0*2:0:0       same as 1*2:0:0

*0:2:0:0       Feb 1 of the current year

 0:0:0*0       same as 0:0:1*0 (every week on
               the first day of the week)

 0:0*0:0       same as 0:1*0:0 (every month
               on the 1st)

 0*0:0:0       same as 1*0:0:0

*0:0:0:0       Jan 1 of the current year
```

METHODS

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

parse

```
$err = $recur->parse($string [,$modifiers] [,$base,$start,$end,$unmod]);
```

This creates a new recurrence. A string containing a valid frequency is required. In addition, `$start`, `$end`, and `$base` dates can be passed in (either as Date::Manip::Date objects, or as strings containing dates that can be parsed), and any number of the modifiers listed above.

If the `$start` or `$end` dates are not included, they may be supplied automatically, based on the value of the RecurRange variable. If any of the dates are passed in, they must be included in the order given (though it is safe to pass an empty string or undef in for any of them if you only want to set some, but not all of them). If `$unmod` is true, the range will apply to unmodified dates rather than the modified dates.

The `$modifiers` argument must contain valid modifiers, or be left out of the argument list entirely. You cannot pass an empty string or undef in for it.

```
$err = $recur->parse($string);
```

This creates a recurrence from a string which contains all of the necessary elements of the recurrence. The string is of the format:

```
FREQ*MODIFIERS*BASE*START*END*UNMOD
```

where FREQ is a string containing a frequency, MODIFIERS is a string containing a comma separated list of modifiers, BASE, START, and END are strings containing parseable dates.

All pieces are optional, but order must be maintained, so all of the following are valid:

```
FREQ*MODIFIERS
FREQ**BASE
FREQ**BASE*START*END
FREQ***START*END*UNMOD
```

If a part of the recurrence is passed in both as part of `$string` and as an argument, the argument overrides the string portion, with the possible exception of modifiers. The modifiers in the argument override the string version unless the first one is a '+' in which case they are appended. See the modifiers method below for more information.

frequency
start
end
basedate
modifiers
You can also create a recurrency in steps (or replace parts of an existing recurrence) using the following:

```
$err = $recur->frequency($frequency);

$err = $recur->start($start);
$err = $recur->start($start,$unmod);
$err = $recur->end($end);

$err = $recur->basedate($base);

$err = $recur->modifiers($modifiers);
$err = $recur->modifiers(@modifiers);
```

These set the appropriate part of the recurrence.

Calling the frequency method discards all information currently stored in the Recur object (including an existing start, end, and base date), so this method should be called first.

In the modifiers method, the modifiers can be passed in as a string containing a comma separated list of modifiers, or as a list of modifiers. The modifiers passed in override all previously set modifiers UNLESS the first one is the string "+", in which case the new modifiers are appended to the list.

In the start, end, and base methods, the date passed in can be a Date::Manip::Date object, or a string that can be parsed to get a date. If `$unmod` is true, it will mean that the range will apply to unmodified dates.

NOTE: the parse method will overwrite all parts of the recurrence, so it is not appropriate to do:

```
$recur->modifiers($modifiers);
$recur->parse($string);
```

The modifiers passed in in the first call will be overwritten.

These functions can also be used to look up the values.

```
$freq  = $recur->frequency();
$start = $recur->start();
$end   = $recur->end();
@mods  = $recur->modifiers();

($base,$actual) = $recur->basedate();
```

The basedate function will return both the specified base and the actual base dates.

If any of the values are not yet determined, nothing will be returned.

dates

```
@dates = $recur->dates([$start,$end,$unmod]);
```

Returns the list of dates defined by the full recurrence. If there is an error, or if there are no dates, an empty list will be returned.

`$start` and `$end` are either `undef`, or dates which can be used to limit the set of dates passed back (they can be Date::Manip::Date objects or strings that can be parsed).

If the recurrence does not have a start and end date already, passing in `$start` and `$end` will set the range (but they will NOT be stored in the recurrence).

If the recurrence does have a start and end date stored in it, the `$start` and `$end` arguments can be used to temporarily override the limits. For example, if a recurrence has a start date of Jan 1, 2006 00:00:00 and and end date of Dec 31, 2006 23:59:59 stored in the recurrence, passing in `$start` of Jul 1, 2006 00:00:00 will limit the dates returned to the range of Jul 1 to Dec 31.

Passing in a start date of Jul 1, 2007 will mean that no dates are returned since the recurrence limits the date to be in 2006.

If one or both of `$start` and `$end` are `undef`, then the stored values will be used.

nth

```
($date,$err) = $recur->nth($n);
```

This returns the `$n`th recurring event (`$n` may be any integer). If an error occurs, it is returned (but it is not set in `$recur` since it may be properly, though perhaps incompletely, defined). The following errors may be returned:

```
Invalid recurrence
   The recurrence has an error flag set.

Incomplete recurrence
   The recurrence is incomplete. It needs either a
   base date or a date range.

Range invalid
   The recurrence has an invalid date range (i.e.
   the end date occurs before the start date).

Start invalid
End invalid
Base invalid
   An invalid date was entered for one of the dates.

Not found
   In some cases, a recurrence may appear valid, but
   does not refer to any actual occurrences.  If no
   dates are found within a certain number of attempts
   (given by the MaxRecurAttempts config variable), this
   error is returned.
```

There are a few special circumstances to be aware of.

1) If the recurrence contains no interval (i.e. is of the form *Y:M:W:D:H:MN:S), the dates come directly from the rtime values. In this case, the 0th event is the first date in the list of dates specified by the rtime. As such, `$n` must be a positive integer. If`$n` is ne gative, or outside the range of dates specified, the returned date will be `undef` (but this is not an error).

2) A very small number of recurrences have an unknown number of recurring events associated with each date. This only happens if one of the values in the rtime is specified as a range including both a positive and negative index. For example, if the day field in an rtime refers to the day of month, and is 15−−15 (i.e. the 15th day to the 15th to the last day), this may include 3 events (on a month with 31 days), 2 event (months with 30 days), 1 event (months with 29 days), or 0 events (months with 28 days). As such, in order to calculate the Nth date, you have to start with the 0th (i.e. base) date and calculate every event until you get the Nth one. For this reason, it is highly recommended that this type of frequency be avoided as it will be quite slow.

3) Most recurrences have a known number of events (equal to the number of combinations of values in the rtime) for each date. For these, calculating the Nth date is much faster. However, in this case, some of them may refer to an invalid date. For example, if the frequency is 'the 31st of every month' and the base (0th) date is Jan 31, the 1st event would refer to Feb 31. Since that isn't valid, `undef` would be returned for `$n=1`. Obviously, it would be possible to actually determine the Nth valid event by calculating all N−1 dates, but in the interest of performance, this is not done.

4) The way the Nth recurring event is calculated differs slightly for NE>0 and N<0 if the delta referred to by the frequency is approximate. To calculate the Nth recurring event (where N>0), you take the base date and add N*DELTA (where DELTA is the delta determined by the frequency). To get the Nth recurring event (where N<0), a date is determine which, if N*DELTA were added to it, would produce

the base date. For more details, refer to the Date::Manip::Calc document. In the "SUBTRACTION" in Date::Manip::Calc section in the discussion of approximate date-delta calculations, calculations are done with `$subtract` = 2.

next
prev

```
($date,$err) = $recur->next();
($date,$err) = $recur->prev();
```

These return the next/previous recurring event.

The first time next/prev is called, one of the recurring events will be selected and returned (using the rules discussed below). Subsequent calls to next/prev will return the next or previous event.

Unlike the **nth** method which will return a specific event (or undef if the Nth even is not defined), the next and prev methods will only work with defined events.

So, for the recurrence:

```
the 31st of every month
```

next might return the following sequence of events:

```
Jan 31 2000
Mar 31 2000
May 31 2000
```

The rules for determining what event to return the first time one of these is called are as follows:

1) If there is a range, next will return the first event that occurs after the start of the range. prev will return the last event that occurs before the end of the range.

2) If there is no range, next will return the first event on or after the base date. prev will return the last event before the base date.

The error codes are the same as for the nth method.

HISTORY OF THE FREQUENCY NOTATION

I realize that the frequency notation described above looks quite complicated at first glance, but it is (IMO) the best notation for expressing recurring events in existence. I actually consider it the single most important contribution to date/time handling in Date::Manip.

When I first decided to add recurring events to Date::Manip, I first came up with a list of common ways of specifying recurring events, and then went looking for a notation that could be used to define them. I was hoping for a notation that would be similar to cron notation, but more powerful.

After looking in several specifications (including ISO 8601) and after a discussion on a mailing list of calendar related topics, it appeared that there was no concise, flexible notation for handling recurring events that would handle all of the common forms I'd come up with.

So, as a matter of necessity, I set about inventing my own notation. As I was looking at my list, it struck me that all of the parts which specified a frequency were higher level (i.e. referred to a larger unit of time) than those parts which specified a specific value (what I've called the rtime). In other words, when the terms were laid out from year down to seconds, the frequency part was always left of specific values.

That led immediately to the notation described above, so I started analyzing it to figure out if it could express all of the recurring events I'd come up with. It succeeded on 100% of them. Not only that, but by playing with different values (especially different combinations of m/w/d values), I found that it would define recurring events that I hadn't even thought of, but which seemed perfectly reasonable in hindsight.

After a very short period, I realized just how powerful this notation was, and set about implementing it, and as I said above, of all the contributions that Date::Manip has made, I consider this to be the most important.

KNOWN BUGS

If you specify a recurrence which cannot be satisfied for the base date, or for any time after the base date, the recurrence will crash. This can only happen if you specify a recurrence that always occurs in the spring DST transition using the current timezone rules.

For example, in a US timezone, the current timezone rules state that a DST transition occurs at 02:00:00 on the 2nd Sunday in March and the clock jumps to 03:00. This started in 2006. As a result, the recurrence

```
1*3:2:7:2:0:0
```

with a base date of 2006 or later cannot be satisfied.

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

This script is free software; you can redistribute it and/or modify it under the same terms as Perl itself.

AUTHOR

Sullivan Beck (sbeck@cpan.org)