NAME

Date::Manip::Delta − Methods for working with deltas

SYNOPSIS

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

DESCRIPTION

This module contains functions useful in parsing and manipulating deltas. As used in this module, the term delta refers to an amount of time elapsed. It includes no information about a starting or ending time.

There are several concepts involved in understanding the properties of a delta.

standard and business delta

There are two different modes for working with deltas: standard and business. The mode used depends on how you treat the calendar.

Standard deltas use the full calendar without any modifications.

A business delta uses a calendar in the way a business might. In a business calendar, anything outside of a business day is ignored. Typically, this includes holidays and weekends. In addition, the part of the day outside of business hours is also ignored, so a day may only run from 08:00 to 17:00 and everything outside of this is ignored.

The length of a work day is usually not 24 hours. It is defined by the start and end of the work day and is set using the config variables: **WorkDayBeg** and **WorkDayEnd** (**WorkDay24Hr** may be used to specify a 24−hour work day). The work week is defined using the config variables: **WorkWeekBeg** and **WorkWeekEnd**.

Daylight saving time are ignored with business calculations because time changes occur at night (usually on the weekends) outside of business hours. This may yield unexpected results if the work day is defined to be 24−hours and the work week includes a day when a time change occurs.

fields

A delta consists of 7 fields: years, months, weeks, days, hours, minutes, and seconds, usually expressed as a colon-separated string. For example:

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

refers to an elapsed amount of time 1 year, 2 months, 3 weeks, 4 days, 5 hours, 6 minutes, and 7 seconds long.

normalized

A delta can be normalized or not. A normalized delta has values which have been simplified based on how a human would think of them. As an example, the delta:

```
0:0:0:0:0:10:70
```

is not normalized since 70 seconds is typically thought of as 1 minute 10 seconds. The normalized form of this delta would be:

```
0:0:0:0:0:11:10
```

By default, deltas are converted to a normalized form in most functions that create/modify a delta, but this can be overridden.

Types of deltas

There are 4 type of deltas that are available.

Exact deltas

The most common type (and the default in most situations) is an exact delta. An exact delta is one where only fields which have exactly known lengths are allowed to be non-zero.

For standard calculations, there are only three exactly known fields (hours, minutes, and seconds). The lengths are defined as:

```
1 hour   = 3600 seconds
1 minute = 60 seconds
```

Note that since a day is NOT always 24 hours (due to daylight saving time changes), a day is not an exactly known field.

For business calculations, a day IS an exactly known field. Since business mode ignores daylight saving time, the length of the day can be calculated based on the config variables listed above. So, for example, if the work day is 08:00−17:00, the length of the day is 9 hours. The length of the week is still unknown since some work weeks may have fewer days than others due to holidays.

All fields which are not exactly known will always have zero value.

Semi-exact deltas

A semi-exact delta treats the day/week fields as if they were exactly known.

For standard calculations, this is done by using the relationships:

```
1 day  = 24 hours
1 week = 7 days
```

For business calculations, it is done by treating a week as a constant length (determined by the config variables listed above) ignoring holidays. So if a typical work week is Mon-Fri, the length of the week is 5 days.

For semi-exact deltas, the value of the year/month must be zero.

Although this may yield some values that are not exactly accurate around daylight saving time transitions, strictly speaking, they yield results that are useful in terms of how humans think of deltas.

Approximate deltas

An approximate delta can have non-zero values for all fields. When normalizing the fields, the year/month fields are treated as one set using the relationship

```
1 year  = 12 months
```

The remaining fields are normalized using the semi-exact relationships.

Estimated deltas

The final type of delta are estimated deltas. These are deltas where an estimated length is applied to all the approximate fields.

For standard deltas, the additional relationship:

```
1 year = 365.2425 days
```

is used. For business deltas, the additional relationship:

```
1 year   = X/7 * 365.2425 days
```

(where X is the number of work days in a week) is used.

Fractional seconds will be discarded (not rounded).

NOTE: it is not possible to look at a delta and determine what type it is. For example, a standard delta with a non-zero day value might be approximate or semi-exact. The type will need to be explicitly selected, or determined by the context of the operation.

signs

Each field has a sign associated with it. For example, the delta ‘‘1 year ago’’ is written as:

```
−1:0:0:0:0:0:0
```

The sign of any field is optional, and if omitted, it is the same as the next higher field. So, the following are identical:

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

In a normalized delta, all fields in a set will have the same sign. So the standard delta:

```
0:0:+3:−2:0:0:0:0   (3 weeks −2 days)
```

is not normalized. The normalized version would be:

```
0:0:+2:5:0:0:0:0    (2 weeks, 5 days)
```

Since an approximate delta has two sets (the y/m set and the w/d/h/mn/s set), these deltas may have two signs. So, the following is a fully normalized approximate delta:

```
+1:0:−3:3:1:0:0
```

fractional values

Fractional fields are allowed such as:

```
1.25 days
1.1 years
```

but whenever parsing a delta with fractional fields, the delta will be normalized using the estimated relationships described above. Fractional seconds will be discarded.

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 = $delta−>parse($string, \%opts);
$err = $delta−>parse($string [,$business] [,$no_normalize]);
```

The second format is supported for backward compatibility, but is deprecated and will be removed in Date::Manip 7.00. The second form is equivalent to:

```
$err = $delta−>parse($string, { business => $business,
                                nonorm   => $no_normalize });
```

This takes a string and parses it to see if it is a valid delta. If it is, an error code of 0 is returned and `$delta` now contains the value of the delta. Otherwise, an error code of 1 is returned and an error condition is set in the delta.

Recognized options are:

```
mode      : standard/business
            to specify if it is a business delta or a standard delta
nonorm    : 0/1
            1 if the delta should not be normalized
type      : exact, semi, approx, estimated
```

When specifying the type, the delta given must satisfy the requirements of the type (i.e. no year field for an exact delta).

A delta string is usually specified in compact notation which consists of a colon separated list of numbers (with optional signs):

```
Examples:
   0:0:0:0:4:3:−2
   +4:3:−2
   +4::3
```

In compact notation, from 1 to 7 of the fields may be given. For example D:H:MN:S may be given to specify only four of the fields. No spaces may be present in the string, but it is allowed to omit some of the fields. For example 5::3:30 is valid. In this case, missing fields default to the value 0.

The delta string may also be specified using common field abbreviations. This is described below in the "ADDITIONAL DELTA NOTATIONS" section.

input

```
$str = $delta−>input();
```

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

set

```
$err = $delta−>set(\%opts);
$err = $delta−>set($field,$val [,$no_normalize]);
```

The second format is supported for backward compatibility, but is deprecated and will be removed in Date::Manip 7.00. The second form is equivalent to:

```
$err = $delta−>set( $field => $val, 'nonorm' => $no_normalize );
```

This explicitly sets one or more parts of a delta. `%opts` is a set of key/value pairs:

```
$key     $val

delta    [Y,M,W,D,H,MN,S]  sets the entire delta
business [Y,M,W,D,H,MN,S]  sets the entire delta
standard [Y,M,W,D,H,MN,S]  sets the entire delta
y        YEAR              sets one field
M        MONTH
w        WEEK
d        DAY
h        HOUR
m        MINUTE
s        SECOND

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

An error is returned if an invalid data is passed in.

`%opts` can only include a single key that affects each field (i.e. you can set **delta** or **business** but not both, and you cannot set both **delta** and **y**, but you CAN set both **y** and **w**).

When setting the entire delta with **business** or **standard**, it flags the delta as a business or standard mode delta respectively. In those cases, you are not allowed to set the **mode** option. Partial deltas are allowed (i.e. [H,MN,S]) in which case zeros are added for all fields not specified.

When setting the entire delta with **delta**, the flag is left unchanged (unless the **mode** option is also passed in).

Also, when setting the entire delta, signs are not carried from one field to another, so [−1,2,...] is equivalent to [−1,+2,...].

By default, a delta is normalized, but setting the **nonorm** key to a true value will not do that.

For backwards compatibility, **normal** can be used in place of **standard**, both as `$field` or as `$val`. This is deprecated and will be removed in Date::Manip 7.00.

When setting any field in the delta, the type of delta will be determined automatically as either **exact** (if only fields that are exactly known are have non-zero fields), **semi** (if only fields that are semi-exact or exact are included), or **approx** otherwise. If the **type** option is set, it will be used provided it is valid (i.e. you cannot set it to **exact** if fields that are not exactly known are set).

printf

```
$out = $delta−>printf($in);
@out = $delta−>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 delta. 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’’.

calc

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

type

```
$flag = $delta−>type($op);
```

This tests to see if a delta is of a certain type. `$op` can be;

```
business  : returns 1 if it is a business delta
standard  : returns 1 if it is a standard (non−business delta)

exact     : returns 1 if it is exact
semi      : returns 1 if it is semi−exact
approx    : returns 1 if it is approximate
estimated : returns 1 if it is estimated
```

value

```
$val = $delta−>value();
@val = $delta−>value();
```

This returns the value of the delta. In scalar context, it returns the printable string (equivalent to the printf directive ’%Dt’). In list context, it returns a list of fields.

An empty string/list is returned if there is no valid delta stored in `$delta`.

convert

```
$delta−>convert($to);
```

This converts a delta from one type to another. `$to` can be ’exact’, ’semi’, or ’approx’. The conversion uses the approximate and estimated relationships listed above to convert the delta.

For example, if the exact non-business delta `$delta` contains:

```
0:0:0:0:44:0:0
```

then the following call:

```
$delta−>convert('semi')
```

would produce the semi-exact delta:

```
0:0:0:1:20:0:0
```

The result will always be normalized.

Converting from one type to another that is less exact (i.e. exact to semi-exact or semi-exact to approx) is supported. Converting the other direction is supported for backward compatibility, but will be removed in 7.00 because that operation is not one that is well defined.

There is currently no support for converting business to non-business (or vice-versa).

cmp

```
$flag = $delta1−>cmp($delta2);
```

This compares two deltas (using the approximate relationships listed above) and returns −1, 0, or 1 which could be used to sort them by length of time.

Both deltas must be valid, and both must be either business or non-business deltas. They do not need to be the same out of exact, semi-exact, and approximate.

undef will be returned if either delta is invalid, or you try to compare a business and non-business delta.

ADDITIONAL DELTA NOTATIONS

When parsing a delta, the string may be specified with the field spelled out, rather than using the colon separated fields.

This expanded notation has the fields spelled out in some language specific form:

```
Examples:
   +4 hours +3mn −2second
   + 4 hr 3 minutes −2
   4 hour + 3 min −2 s
   4 hr 2 s
```

A field in the expanded notation has an optional sign, a number, and a string specifying the type of field. If the sign is absent, it defaults to the sign of the next larger element. So the following are equivalent:

```
−4 hr 3 min 2 sec
−4 hr −3 min −2 sec
```

The valid strings describing each of the fields is contained in ‘‘Delta field names’’ section of the appropriate Date::Manip::Lang::<LANGUAGE> document. Refer to the Date::Manip::Lang document for a list of languages.

For example, for English, the document is Date::Manip::Lang::English and the field names include strings like:

```
y:  y, yr, year, years
m:  m, mon, mons, month, months
w:  w, wk, ws, wks, week, weeks
d:  d, day, days
h:  h, hr, hrs, hour, hours
mn: mn, min, mins, minute, minutes
s:  s, sec, secs, second, seconds
```

This list may not be complete. You should refer to the language document for the full list.

The ‘‘seconds’’ string may be omitted. The sign, number, and string may all be separated from each other by any amount of whitespace. The string specifying the unit must be separated from a following number by whitespace or a comma, so the following example will NOT work:

```
4hours3minutes
```

At minimum, it must be expressed as:

```
   4hours 3minutes
   4 hours, 3 minutes
```

In the the expanded format, all fields must be given in the order: Y M W D H MN S. Any number of them may be omitted provided the rest remain in the correct order. Small numbers may be spelled out, so

```
   in two weeks
   in 2 weeks
```

both work (but do not rely on this to work for large numbers).

Most languages also allow a word to specify whether the delta is an amount of time after or before a fixed point. In English, the word ‘‘in’’ refers to a time after a fixed point, and ‘‘ago’’ refers to a point before a fixed point. So, the following deltas are equivalent:

```
  1:0:0:0:0:0:0
  in 1 year
```

and the following are equivalent

```
  −1:0:0:0:0:0:0
  1 year ago
```

The word ‘‘in’’ is completely ignored. The word ‘‘ago’’ has the affect of reversing all signs that appear in front of the components of the delta. In other words, the following two strings are identical:

```
   −12 yr  6 mon ago
   +12 yr +6 mon
```

(don’t forget that there is an implied minus sign in front of the 6 in the first string because when no sign is explicitly given, it carries the previously entered sign).

The in/ago words only apply to the expanded format, so the following is invalid:

```
   1:0:0 ago
```

A delta may be standard (non-business) or business. By default, a delta is treated as a non-business delta, but this can be changed in two different ways.

The first way to make a delta be business is to pass in the appropriate option. For example:

```
  $delta−>parse($string, { 'mode' => 'business' });
  $delta−>parse($string, { 'mode' => 'standard' });
```

The second way to specify whether a delta is business or non-business is to include a key word in the string that is parsed. If this string is included, it should not conflict with the value of a ’mode’ option.

Most languages include a word like ‘‘business’’ which can be used to specify that the resulting delta is a business delta or a non-business delta. Other languages have equivalent words. The placement of the word is not important. Also, the ‘‘business’’ word can be included with all types of deltas, and in both compact and expanded notation, so the following are valid and equivalent:

```
   in 4 hours business
   4:0:0 business
   business 0:0:0:0:4:0:0
```

There are also words ‘‘exact’’ or ‘‘approximate’’ which may be included in the delta for backward compatibility. However, they will be ignored. They will be removed in Date::Manip 7.00. The accuracy of delta (exact, semi-exact, approximate) will be determined only by what fields are present in the delta and the options passed in. When a delta is parsed, it is automatically normalized, unless the ’nonorm’ option is passed in.

PRINTF DIRECTIVES

The following printf directives are replaced with information from the delta. Directives may be replaced by the values of a single field in the delta (i.e. the hours or weeks field), the value of several fields expressed in terms of one of them (i.e. the number of years and months expressed in terms of months), or the directive may format either the entire delta, or portions of it.

Simple directives

These are directives which print simple characters. Currently, the only one is:

```
%%    Replaced by a single '%'
```

As an example:

```
$delta−>printf('|%%|');
   => |%|
```

Directives to print out a single field

The following directive is used to print out the value of a single field. Spaces are included here for clarity, but are not in the actual directive.

```
% [+] [pad] [width] Xv
```

Here, X is one of (y,M,w,d,h,m,s). The directive will print out the value for that field.

If a ’+’ is included immediately after the ’%’, a sign will always be included. By default, only negative values will include a sign.

’width’ and ’pad’ are used to set the width of the string containing the field as well as how it is padded.

’width’ is any positive integer (without a sign). If ’width’ is included, it sets the length of the output string (unless the string is already longer than that, in which case the ’width’ is ignored).

If ’pad’ is included, it may be the character ’<’, ’>’, or ’0’. It will be ignored if ’width’ is not included, or the string is already longer than ’width’. If the formatted delta field is shorter than ’width’, it will be padded with spaces on the left (if ’pad’ is ’<’), or right (if ’pad’ is ’>’), or it will be padded on the left (after any sign) with zeroes (if ’pad’ is ’0’).

In the following examples, `$delta` contains the delta: 1:2:3:4:5:6:7

```
$delta−>printf('|Month: %Mv|');
   => |Month: 2|

$delta−>printf('|Day: %+05dv|');
   => |Day: +0004|

$delta−>printf('|Day: %+<5dv|');
   => |Day:    +4|

$delta−>printf('|Day: %>5sv|');
   => |Day: 7    |
```

Directives to print out several fields in terms of one of them

The following directive is used to print out the value of several different fields, expressed in terms of a single field.

```
% [+] [pad] [width] [.precision] XYZ
```

Here, X, Y, and Z are each one of (y,M,w,d,h,m,s). The directive will print out the value for fields Y through Z expressed in terms of field X.

Y must come before Z in the sequence (y,M,w,d,h,m,s) or it can be the same as Z.

So, to print the day and hour fields in terms of seconds, use the directive:

```
%sdh
```

Any time all of X, Y, and Z are from a single set of fields, exact relationships are used.

If the X, Y, and Z fields do not all belong to the same set of fields, approximate relationships are used.

For non-business deltas, an approximate relationship is needed to link the Y/M part of the delta to the W/D part and a semi-approximate relationship is needed to link the W/D part with the H/MN/S part. These relationships are:

```
1 day    = 24 hours
1 year   = 365.2425
```

For business deltas, the approximate and semi-approximate relationships used to link the fields together are:

```
1 week   = X    (length of business week in days)
1 year   = X/7 * 365.2425
```

For business deltas, the length of the day is defined using WorkDayStart and WorkDayEnd. For non-business deltas, a day is 24 hours long (i.e. daylight saving time is ignored).

If ’precision’ is included, it is the number of decimal places to print. If it is not included, but ’width’ is included, precision will be set automatically to display the maximum number of decimal places given ’width’.

If ’pad’ is included, it may be the character ’<’, ’>’, or ’0’, and is used in the same way as printing out a single field.

In the following examples, `$delta` contains the delta: 1:2:3:4:5:6:7

```
$delta−>printf('|%.4Myw|');
   => |14.6900|
   1 year, 2 months, 3 weeks is approximately
   14.6900 months
```

Directives to print out portions of the delta

The following directives may be used to print out some or all of a delta.

```
% [+] [pad] [width] Dt
% [+] [pad] [width] DXY
```

The first directive will print out the entire delta.

The second will print out the delta from the X to Y fields inclusive (where X and Y are each one of (y,M,w,d,h,m,s) and X must come before Y in the sequence).

’pad’ is optional and can be either ’<’ or ’>’ meaning to pad on the left or right with spaces. It defaults to ’<’.

If a ’+’ is included immediately following the ’%’, every field will have a sign attached. Otherwise, only the leftmost field in each set of fields will include a sign.

```
 $delta−>printf('|%Dt|');
    => |+1:2:+3:+4:5:6:7|

 $delta−>printf('|%+Dyd|');
    => |+1:+2:+3:+4|
```

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)