NAME

Date::Manip::Config − Date::Manip configuration

SYNOPSIS

This documents the configuration information which is stored in each Date::Manip::Base object, how to modify this information, and how the information is used in the other Date::Manip modules.

DESCRIPTION

Date::Manip is a very configurable bundle of modules. Many of it's behaviors can be modified to change how date operations are done. To do this, a list of configuration variables may be set which define many Date::Manip behaviors.

There are three ways to set config variables. The first two are to pass them in when creating an object, or to pass them to the config method after the object is created. All of the main Date::Manip modules (Date::Manip::Base, Date::Manip::TZ, Date::Manip::Date, Date::Manip::Delta, and Date::Manip::Recur) have the config method.

As an example, you can create and configure a Date::Manip::Date object using the commands:

```
$date = new Date::Manip::Date;
$date−>config($var1,$val1,$var2,$val2,...);
```

This can be shortened to:

```
$date = new Date::Manip::Date [$var1,$val1,...];
```

The values of the config variables are stored in the Date::Manip::Base object. So, if you have a Date::Manip::Date object, it has a Date::Manip::Base object associated with it, and the configuration information is stored there. The same Date::Manip::Base object may be used by any number of higher objects, and all will share the same configuration. If multiple Date::Manip::Date objects share the same Date::Manip::Base object, setting a configuration variable on any of them affects all of the Date::Manip::Date objects. If you need to work with different configurations simultaneously, it is necessary to work with multiple Date::Manip::Base objects. This is covered in the Date::Manip::Objects document.

An alternate method exists if you are using one of the functional interfaces. To set a variable using the functional interface, use the call:

```
Date_Init("$var1=$val1");
```

The third way to set config variables is to store them in a config file. The config file is read in by passing the appropriate values to the config method as described below. A config file is a good way to easily change a large number of settings. They are also necessary for other purposes (such as events and holidays which are covered in the Date::Manip::Holidays document).

CONFIG FILES

One of the variables that can be passed to the config method is ‘‘ConfigFile’’. The value of this variable is the path to a config file.

When any Date::Manip::* object is configured, any number of config files may be read (and the config files can specify additional files to read).

The starting section of a config file contains general configuration variables. A list of all config variables is given below.

Following this, any number of special sections may be included in the config file. The special sections are used to specify other types of information, such as a list of holidays or special events. These special sections are described elsewhere in the documentation.

The syntax of the config file is very simple. Every line is of the form:

```
VAR = VAL
```

or

```
*SECTION
```

Blank lines and lines beginning with a pound sign (#) are ignored. All whitespace is optional. Variables names in the main section and section names are case insensitive (though values in the main section are typically case sensitive). Strings in other sections (both variables and values) are case sensitive.

The following is a sample config file:

```
DateFormat = US
Language   = English

*Holidays

Dec 25 =  Christmas
Jan 1  =  New Year's
```

All config variables that may appear in the main part of a config file are described in the next section. Other sections are described elsewhere. The *Holidays and *Events sections are both described in the Date::Manip::Holidays documentation.

A sample config file is included with the Date::Manip distribution. Modify it as appropriate and copy it to some appropriate directory and use the ConfigFile variable to access it. For example, if a config file is stored in */home/foo/Manip.cnf*, you can load it by:

```
$date−>config("ConfigFile","/home/foo/Manip.cnf");
```

or (if using a functional interface):

```
Date_Init("ConfigFile=/home/foo/Manip.cnf");
```

NOTE: if you use business mode calculations, you must have a config file since this is the only place where you can define holidays.

In the top section, only variables described below may be used. In other sections, checking (if any) is done in the module that uses the data from that section.

BASIC CONFIGURATION VARIABLES

This section describes the basic Date::Manip configuration variables which can be used in a config file, or which may be passed in using the appropriate functions for each module.

Variable names are case insensitive, both as arguments to the config function and in the config file. The values are case sensitive except where specified otherwise.

Defaults

The value for this config variable is ignored. Whenever the Defaults config variable is encountered, the defaults for all config variables are restored, overriding ALL changes that have been made.

In other words, in the following call:

```
$date−>config("Language","Russian",
              "Defaults","1");
```

the first option will end up being ignored since the Defaults config variable will set the language back to it’s default value which is English.

When using a functional interface, use:

```
Date_Init("Defaults=1");
```

ConfigFile

The ConfigFile variable defines a config file which will be parsed for configuration information. It may be included any number of times, each one including the path to a single config file. The value of this variable is a full path to a file.

An example call to the config function might be:

```
$date->config("ConfigFile","/tmp/file1",
              'Language',$val);
```

Config files are parsed immediately when encountered. So in this example, the file */tmp/file1* will be parsed before the next variable ('Language'). In addition, if a config file contains a ConfigFile variable, that file will immediately be parsed before continuing with the original file.

The path to the file may be specified in any way valid for the operating system. If a file is not found, a warning will be issued, but execution will continue.

Multiple config files are safe, and a section may safely be split across multiple files.

When using a functional interface, use:

```
Date_Init("ConfigFile=/tmp/file1");
```

Language

Date::Manip can be used to parse dates in many different languages. A list of the languages is given in the Date::Manip::Lang document.

To parse dates in a different language, just use the Language config variable with the name of the language as the value. Language names are case insensitive.

Additional languages may be added with the help of someone fluent in English and the other language. If you are interested in providing a translation for a new language, please refer to the Date::Manip::Lang document for instructions.

Encoding

Date::Manip has some support for handling date strings encoded in alternate character encodings.

By default, input strings may be tested using multiple encodings that are commonly used for the specific languages, as well as using standard perl escape sequences, and output is done in UTF−8.

The input, output, or both can be overridden using the Encoding variable.

Setting Encoding to the name of a single encoding (a name supported by the Encoding perl module), will force all input and output to be done in that encoding.

So, setting:

```
Encoding = iso−8859−1
```

means that all input and output will be in that encoding. The encoding 'perl' has the special meaning of storing the string in perl escape sequences.

Encoding can also be set to the name of two encoding (separated by a comma).

```
Encoding = iso−8859−1,utf−16
```

which means that all input is in iso−8859−1 encoding, but all output will be utf−16.

Encoding may also be set as follows:

```
Encoding = iso−8859−1,
```

meaning that input is in iso−8859−1 and output is in the default (i.e. UTF−8) encoding.

```
Encoding = ,utf−16
```

means to check the input in all of the encodings, but all output will be in utf−16 encoding.

Note that any time you change languages, it will reset the encodings, so you should set this config variable AFTER setting the language.

FirstDay

It is sometimes necessary to know what day of week is regarded as first. By default, this is set to Monday as that conforms to ISO 8601, but many countries and people will prefer Sunday (and in a few cases, a different day may be desired). Set the FirstDay variable to be the first day of the week (1=Monday, 7=Sunday).

Jan1Week1

ISO 8601 states that the first week of the year is the one which contains Jan 4 (i.e. it is the first week in which most of the days in that week fall in that year). This means that the first 3 days of the year may be treated as belonging to the last week of the previous year. If this is set to non-nil, the ISO 8601 standard will be ignored and the first week of the year contains Jan 1.

Printable

Some commands may produce a printable version of a date. By default, the printable version of the date is of the format:

```
    YYYYMMDDHH:MN:SS
```

Two other simple versions have been created. If the Printable variable is set to 1, the format is:

```
    YYYYMMDDHHMNSS
```

If Printable is set to 2, the format is:

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

This config variable is present in order to maintain backward compatibility, and may actually be deprecated at some point. As such, additional formats will not be added. Instead, use the printf method in the Date::Manip::Date module to extract information with complete flexibility.

DATE PARSING CONFIGURATION VARIABLES

DateFormat

Different countries look at the date 12/10 as Dec 10 or Oct 12. In the United States, the first is most common, but this certainly doesn't hold true for other countries. Setting DateFormat to ''US'' (case insensitive) forces the first behavior (Dec 10). Setting DateFormat to anything else forces the second behavior (Oct 12). The ''US'' setting is the default (sorry about that... I live in the US).

YYtoYYYY

When parsing a date containing a 2−digit year, the year must be converted to 4 digits. This config variable determines how this is done.

By default, a 2 digit year is treated as falling in the 100 year period of CURR−89 to CURR+10. So in the year 2005, a two digit year will be somewhere in the range 1916 to 2015.

YYtoYYYY may be set to any integer N to force a 2 digit year into the period CURR-N to CURR+(99−N). A value of 0 forces the year to be the current year or later. A value of 99 forces the year to be the current year or earlier. Although the most common choice of values will be somewhere between 0 and 99, there is no restriction on N that forces it to be so. It can actually be any positive or negative number you want to force it into any 100 year period desired.

YYtoYYYY can also be set to ''C'' to force it into the current century, or to ''C##'' to force it into a specific century. So, in 1998, ''C'' forces 2 digit years to be 1900−1999. ''C18'' would always force a 2 digit year to be in the range 1800−1899. Note: I'm aware that the actual definitions of century are 1901−2000, NOT 1900−1999, so for purists, treat this as the way to supply the first two digits rather than as supplying a century.

It can also be set to the form ''C####'' to force it into a specific 100 year period. C1950 refers to 1950−2049.

DefaultTime

When a date is parsed from one of the formats listed in the ''Common date formats'' or ''Less common formats'' sections of the Date::Manip::Date document, and no time is explicitly included, the default time can be determined by the value of this variable. The two possible values are:

```
   midnight   the default time is 00:00:00
   curr       the default time is the current time
```

‘‘midnight’’ is the default value.

NOTE: this only applies to dates parsed with the parse method. Dates parsed using the parse_date method always default to 00:00:00.

PeriodTimeSep

By default, the time separator (i.e. the character that separates hours from minutes and minutes from seconds) is specified in the language translations and in most cases it does not include a period. In English, the only defined time separator is a colon (:), so the time can be written as 12:15:30 .

If you want to use a period (.) as a time separator as well, set this to 1. Then you can write the time as 12.15.30 .

By default, a period is used as a date separator, so 12.15.30 would be interpreted as Dec 15 1930 (or 2030), so if you use the period as a date separator, it should not be used as a time separator too.

Format_MMMYYYY

By default, when parsing a string like ’Jun 1925’, it will be interpreted as ’Jun 19, 2025’ (i.e. MMM DDYY). Also, the string ’1925 Jun’ is not allowed.

This variable can be set to either ’first’ or ’last’, and in that case, both ’Jun 1925’ and ’1925 Jun’ will be allowed, and will refer to either the first or last day of June in 1925.

BUSINESS CONFIGURATION VARIABLES

These are configuration variables used to define work days and holidays used in business mode calculations. Refer to the Date::Manip::Calc documentation for details on these calculations.

WorkWeekBeg
WorkWeekEnd

The first and last days of the work week. These default to Monday and Friday. Days are numbered from 1 (Monday) to 7 (Sunday). WorkWeekBeg must come before WorkWeekEnd numerically so there is no way to handle a work week of Sunday to Thursday using these variables.

There is also no way to handle an odd work schedule such as 10 days on, 4 days off.

However, both of these situations can be handled using a fairly simple workaround.

To handle a work week of Sunday to Thursday, just set WorkWeekBeg=1 and WorkWeekEnd=7 and defined a holiday that occurs every Friday and Saturday.

To handle a 10 days on, 4 days off schedule, do something similar but defined a holiday that occurs on all of the 4 days off.

Both of these can be done using recurrences. Refer to the Date::Manip::Recur documentation for details.

WorkDay24Hr
WorkDayBeg
WorkDayEnd

If WorkDay24Hr is non-zero, a work day is treated as usually being 24 hours long (daylight saving time changes ARE taken into account). The WorkDayBeg and WorkDayEnd variables are ignored in this case.

By default, WorkDay24Hr is zero, and the work day is defined by the WorkDayBeg and WorkDayEnd variables. These are the times when the work day starts and ends respectively. WorkDayBeg must come before WorkDayEnd (i.e. there is no way to handle the night shift where the work day starts one day and ends another).

The time in both should be a valid time format (H, H:M, or H:M:S).

Note that setting WorkDay24Hr to a non-zero value automatically sets WorkDayBeg and

WorkDayEnd to ''00:00:00'' and ''24:00:00'' respectively, so to switch back to a non−24 hour day, you will need to reset both of those config variables.

Similarly, setting either the WorkDayBeg or WorkDayEnd variables automatically turns off WorkDay24Hr.

TomorrowFirst

Periodically, if a day is not a business day, we need to find the nearest business day to it. By default, we'll look to ''tomorrow'' first, but if this variable is set to 0, we'll look to ''yesterday'' first. This is only used in the `Date::Manip::Date::nearest_business_day` method (and the `Date_NearestWorkDay` function) and is easily overridden (see documentation for the nearest_business_day method).

EraseHolidays
EraseEvents

If these variables are used (a value must be passed in, but is ignored), the current list of defined holidays or events is erased. A new set will be set the next time a config file is read in.

Although these variables are supported, the best way to have multiple holiday or events lists will be to create multiple Date::Manip::Base objects based on separate config files.

RECURRENCE CONFIGURATION VARIABLES

The following config variables help in the handling of recurrences.

RecurRange

When a recurrence is created, it begins with a default range (start and end date). The range selected depends on the value of this variable, and can be set to any of the following:

```
none     no default range supplied
year     the current year
month    the current month
week     the current week
day      the current day
all      Jan 2, 0001 to Dec 30, 9999
```

The default value is ''none''.

MaxRecurAttempts

Occasionally, a recurrence is invalid (though it cannot be easily determined in advance).

When searching for dates that match the recurrence, this is the maximum number of attempts that will be done. If none are found, the recurrence will be assumed to be invalid.

TIME ZONE RELATED CONFIGURATION VARIABLES

The following configuration variables may alter the current time zone. As such, they are only available once the Date::Manip::TZ module is available. An easy way to handle this is to only pass them to the config method of a Date::Manip::TZ object or one of the high level objects (Date::Manip::Date, Date::Manip::Delta, or Date::Manip::Recur).

Many of Date::Manip's operations rely on knowing what time it is now. This consists of three things: knowing what date and time it is, knowing what time zone it is, and knowing whether it is daylight saving or not. All of this is necessary in order to correctly handle every possible date.

The daylight saving time information is only used for a couple hours each year during daylight saving time changes (at all other times, the date, time, and time zone are sufficient information), so it is optional, and defaults to standard time if omitted.

The default behavior of Date::Manip is to use the system localtime function to determine the date, time, and daylight saving time information, and to use various methods (see ''DETERMINING THE SYSTEM TIME ZONE'' in Date::Manip::TZ) to determine what time zone the computer is in.

TZ This variable is deprecated, but will be supported for several releases. The SetDate or ForceDate variables (described next) should be used instead.

The following are equivalent:

```
$date−>config("tz","Europe/Rome");
$date−>config("setdate","now,Europe/Rome");
```

or in the functional interface:

```
Date_Init("tz=Europe/Rome");
Date_Init("setdate=now,Europe/Rome");
```

SetDate

The SetDate config variable is used to set the current date, time, or time zone, but then allow it to change over time using the rules of that time zone.

There are several cases where this may be useful.

Often, you may want to use the system time to get the date and time, but you want to work in another time zone. For this, use the call:

```
$date−>config("setdate","now,ZONE");
```

or in the function interface:

```
Date_Init("setdate=now,ZONE");
```

If it is currently

```
Jun 6, 2009 12:00:00 in the America/New_York time zone
```

and you call:

```
$date−>config("setdate","now,Europe/Rome");
```

the Date::Manip will treat that exact instant as

```
Jun 6, 2009 12:00:00 in the Europe/Rome time zone
```

At that precise moment, looking at the system time and parsing the date ‘‘now’’ in Date::Manip will give the same date and time.

The time will continue to advance, but it will use time change rules from the Europe/Rome time zone. What that means is that if a daylight saving time occurs on the computer, but NOT in the Europe/Rome time zone (or vice versa), the system date and time will no longer match the results of parsing the date ‘‘now’’ in Date::Manip.

In general (unless the program runs for an extended period of time), the system date and time WILL match the value of ‘‘now’’, so this is a good way to simulate placing the computer in another time zone.

If the current date/time is ambiguous (i.e. it exists in both standard and daylight saving time in the alternate zone), you can use the call:

```
$date−>config("setdate","now,DSTFLAG,ZONE");
```

to force it to be in one or the other. DSTFLAG can be ‘‘std’’, ‘‘dst’’, ‘‘stdonly’’, or ‘‘dstonly’’. ‘‘std’’ and ‘‘dst’’ mean that the date can be in either standard or saving time, but will try standard first (for ‘‘dst’’) or saving time first (if ‘‘dst’’), and will only try the other if the date is not valid. If ‘‘stdonly’’ or ‘‘dstonly’’ is used, the date will be forced to be standard or saving time respectively (an error will be triggered if there is no valid date in that time).

If the current date/time doesn’t exist in the alternate zone, an error will occur.

The other common operation is that you might want to see results as they would appear on a computer running in a different time zone.

This can be done using the call:

```
$date−>config("setdate","zone,ZONE");
$date−>config("setdate","zone,DSTFLAG,ZONE");
```

If it is currently

```
Jun 6, 2009 12:00:00 in the America/New_York time zone
```

and you call:

```
$date−>config("setdate","zone,America/Chicago");
```

then parsing ‘‘now’’ at precisely that moment will return ‘‘Jun 6, 2009 11:00:00’’. This is equivalent to working in the current zone, but then converting everything to the alternate zone.

Note that DSTFLAG is only used if ZONE is entered as an offset.

The final case where the SetDate config variable is used is to alter the date and time to some other value (completely independent of the current date and time) and allow it to advance normally from that point.

```
$date−>config("setdate","DATE");
$date−>config("setdate","DATE,ZONE");
$date−>config("setdate","DATE,DSTFLAG,ZONE");
```

set both the date/time and zone.

If DATE is not valid in the time zone (either the local time zone or the specified one), and error occurs.

The call:

```
$date−>config("setdate","now");
```

resets everything to use the current date/time and zone and lets it advance normally.

ForceDate

The ForceDate config variable is similar to the SetDate variable, except that once ‘‘now’’ is set, it is not allowed to change. Parsing the date ‘‘now’’ will not change, regardless of how long the program runs (unless either the SetDate or ForceDate variables are set to some other value).

```
$date−>config("forcedate","now,ZONE");
$date−>config("forcedate","now,DSTFLAG,ZONE");
$date−>config("forcedate","zone,ZONE");
$date−>config("forcedate","zone,DSTFLAG,ZONE");
$date−>config("forcedate","DATE");
$date−>config("forcedate","DATE,ZONE");
$date−>config("forcedate","DATE,DSTFLAG,ZONE");
$date−>config("forcedate","now");
```

all set ‘‘now’’ in the same way as the SetDate variable. Spaces after commas are ignored.

ZONE can be any time zone name, alias, abbreviation, or offset, and the best time zone will be determined from all given information.

It should be noted that setting the SetDate or ForceDate variable twice will always refer to the system date/time as a starting point. For example, if a program is running, and calls the method:

```
$date−>config("forcedate","now");
```

at Jun 6, 2009 at 12:00, that time will be treated as now from that point on. If the same call is done an hour later, ‘‘now’’ will then be Jun 6, 2009 at 13:00 from that moment on.

Since the current date is used in the date parsing routines, no parsing can be done on the DATE value in any of the calls. Instead, DATE must be a date in one of the two formats:

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

DEPRECATED CONFIGURATION VARIABLES

The following config variables are currently supported, but are deprecated. They will be removed in a future Date::Manip release:

TZ This is discussed above. Use SetDate or ForceDate instead.

Scheduled for removal 2016−03−01

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

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)