NAME

Date::Manip::TZ − an interface to the time zone data

SYNOPSIS

```
   use Date::Manip::TZ;
   $tz = new Date::Manip::TZ;
```

Data for most (and hopefully all) time zones used around the world have been gathered and is publicly available in the zoneinfo (or Olson) database.

This module uses the data from the zoneinfo database to perform various time zone operations.

DESCRIPTION

Every time zone has some of the following characteristics:

name

Every time zone has a unique name. In the zoneinfo database, these are something similar to:

```
   America/New_York
```

aliases

Time zones may have (but are not required to have) one or more aliases. Each alias is unique, and is not the same as any time zone name. An alias can be used in exactly the same way as a name.

periods

Every time zone is broken up into periods. Each period describes how a portion of time relates to GMT, and how it might be expressed.

Each period includes the following information:

start time, end time

The period begin and ends at certain times. The times are included both as an absolute GMT time, and as a wall clock time. The wall clock start time is the time that will be on a clock just as the period starts (i.e. after a time change). The wall clock end time is the time on a clock immediately before the period ends.

offset

The entire period has an offset which is how much the wall clock time differs from GMT.

abbreviation

When expressing the time period, an abbreviation (such as EST) is typically used.

daylight saving time flag

Every period is categorized as a standard time or a daylight saving time. The flag will be 1 if it is a daylight saving time, or 0 if it is a standard time.

Date::Manip includes all of the data for all of the time zones from the zoneinfo database. This data is available from:

<ftp://ftp.iana.org/tz/>

Additional data from other standards are also used.

The zoneinfo database is not necessary in order to use Date::Manip. Instead, all of that data has been extracted and stored in a series of other modules which are used to handle each time zone. In that way, Date::Manip has no dependency on any other source of data.

The Date::Manip::Zones document contains detailed information on the data available.

METHODS

In all methods, the following variables are used:

`$zone`

This is a string which contains a valid time zone name. For example:

```
  America/New_York
```

$alias

This is a strings which contains a valid time zone name, or a valid time zone alias. For example:

```
America/New_York
US/Eastern
EST5EDT
```

$abbrev

This is a string which contains a valid time zone abbreviation. For example:

```
EST
```

$offset

This is a time zone entered as an offset. An offset is either a string of one of the formats:

```
+HH
+HHMM
+HHMMSS
+HH:MM
+HH:MM:SS
```

or it can be a list reference:

```
[HH,MM,SS]
```

If a list reference is used, the sign must be included with all values. So, the offset "−05:30" would be the list reference:

```
[−5,−30,0]
```

$dstflag

This is always one of the values: std, stdonly, dst, dstonly

It defaults to "std" if it is not present. When determining a time zone, it is usually necessary to check a number of different time zone and DST combinations.

If `$dstflag` is "std", it will check both standard and daylight saving times, but will give preference to standard times. If `$dstflag` is "stdonly", only standard times will be checked.

The "dst" flag will search both, but give preference to daylight saving times. The "dstonly" values will only use daylight saving times.

$date

A date is always a string containing a date in one of the formats:

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

or a list reference:

```
[Y,M,D,H,MN,S]
```

$isdst

This is 0 if a date is in standard time, 1 if it is in daylight saving time.

$period

A period is a list reference currently containing the following items:

```
[ $dateUT, $dateLT, $offsetstr, $offset, $abbrev, $isdst,
  $endUT, $endLT, $begUTs, $begLTs, $endUTs, $endLTs ]
```

`$dateUT` and `$dateLT` are the starting date of the period (i.e. the first second in a period) in universal (GMT) time and local (wall clock) time. `$endUT` and `$endLT` are the end date of the period (i.e. the last second in a period) in universal and local time. These are all stored as list references.

`$offsetstr` is the string representation of the offset ("+05:00:00") and `$offset` is the corresponding list reference form ([5,0,0]).

`$abbrev` is the abbreviation that applies during this period, and `$isdst` is 0 or 1 if it is standard or daylight saving time.

When accessing the elements in a period, use ONLY positive indices. In other words, to get `$endUT`, access it as `$$period[6]`, NOT as `$$period[-2]`, since I am considering adding more information to the period description that may speed up performance.

`$begUTs` is the string representation (YYYYMMDDHH:MN:SS) of `$begUT`. Similar for`$begLTs`, `$endUTs`, and `$endLTs`.

The following methods are available:

base
config
err
new
new_config

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

all_periods

```
   @periods = $tz->all_periods($zone,$year);
```

This returns the description of all time zone periods that occur (in full or in part) during the given year. The year is measured in universal (GMT) time.

convert
convert_to_gmt
convert_from_gmt
convert_to_local
convert_from_local

These functions convert a date from one time zone to another.

```
   ($err,$date,$offset,$isdst,$abbrev) =
      $tz->convert($date,$from,$to [,$isdst]);
```

This converts a date in the time zone given by `$from` to the time zone given by `$to`.

```
   ($err,$date,$offset,$isdst,$abbrev) =
      $tz->convert_to_gmt($date [,$from] [,$isdst]);
```

This converts a date to GMT. If `$from` is given, it is the current time zone of the date. If `$from` is omitted, it defaults to the local time zone.

The value of `$isdst` returned is always 0.

```
   ($err,$date,$offset,$isdst,$abbrev) =
      $tz->convert_from_gmt($date [,$to]);
```

This converts a date from GMT to another time zone. If `$to` is given, the date is converted to that time zone. Otherwise, it is converted to the local time zone.

```
   ($err,$date,$offset,$isdst,$abbrev) =
      $tz->convert_to_local($date [,$from] [,$isdst]);
   ($err,$date,$offset,$isdst,$abbrev) =
      $tz->convert_from_local($date [,$to] [,$isdst]);
```

Similar to the **convert_to_gmt** and **convert_from_gmt** functions. If `$from` or `$to` are omitted, they default to GMT.

If there is any ambiguity about whether `$date` is in DST or not (i.e. if it is a date that is repeated during a time change due to the clock being moved back), the `$isdst` option can be passed in as an argument (it should be 0 or 1) to say which time to use. It is ignored in all cases where `$date` can be determined without that information.

The `$isdst` value passed back is 1 if the converted date is in DST. The `$offset` value passed back is a list reference containing the offset from GMT. `$abbrev` passed back is the time zone abbreviation.

Error codes are:

```
0  No error
1  Invalid arguments
2  Invalid FROM zone
3  Invalid TO zone
4  Invalid date
```

curr_zone

```
$tz->curr_zone();
```

This returns the system time zone. The system time zone is determined using the methods described below in the "DETERMINING THE SYSTEM TIME ZONE" section.

This is the time zone that is used by default unless the SetDate or ForceDate config variable is set to a different zone.

```
$tz->curr_zone(1);
```

This clears the system time zone and re-determines it using the methods described below.

The main reason to do this is if the curr_zone_methods method is used to change how the time zone is determined.

curr_zone_methods

```
$tz->curr_zone_methods(@methods);
```

This sets the list and order of methods to use in determining the local time zone. The various methods available are listed below in the section "DETERMINING THE SYSTEM TIME ZONE".

Some methods may require one or more arguments. For example, the method named "mainvar" takes an option that is the name of a variable. The arguments must be included in the `@methods` list immediately after the method name (so `@methods` is actually a mixture of method names and arguments).

This method may not be used in any environment where taint checking is enabled. If it is, it will issue a warning, but will NOT change the method list.

date_period

```
$period = $tz->date_period($date,$zone,$wall_clock [,$isdst]);
```

This returns the period information for the given date. `$date` defaults to GMT, but may be given as local (i.e. wall clock) time if `$wall_clock` is non-zero. The period information is described in the periods method below.

If a wall clock time is given, no period is returned if the wall clock time doesn't ever appear (such as when a time change results in the clock moving forward "skipping" a period of time). If the wall clock time appears twice (i.e. when a time change results in the clock being set back), the `$isdst` variable is used. The standard time is used unless `$isdst` is non-zero. `$isdst` is ignored except in the case where there are two possible periods.

define_abbrev

```
($err,$val) = $tz->define_abbrev($abbrev,@zone);
```

When encountering an abbreviation, by default, all time zones which ever include the abbreviation will be examine in the order given in the Date::Manip::Zones manual.

Occasionally, it may be necessary to change the order. This is true if you are parsing dates in a time zone which uses an abbreviation which is also used in another time zone, and where the other time zone is given preference. As an example, the abbreviation "ADT" will default to the "Atlantic/Bermuda" time zone. If you are in the "America/Halifax" time zone (which also uses that abbreviation), you may want to change the order of time zones.

This will take an abbreviation (which must be a known abbreviation... there is no means of defining a totally new abbreviation) and a list of zones. This will set the list of zones that will be checked, and the order in which they are checked, when a date is encountered with the given abbreviation. It is not necessary that the list include every zone that has ever used the abbreviation, but it may not include a zone that has never used it.

If `$abbrev` is "reset", all abbreviations are reset to the standard values. If`@zone` includes only the element 'reset', the default list for `$abbrev` is restored.

The following error codes are returned:

```
0  No error
1  $abbrev is not a valid abbreviation in any time zone
2  A zone (returned as $val) is not a valid time zone
3  A zone (returned as $val) does not use the abbreviation
```

For more information about the different zones which may correspond to each abbreviation, and the order in which they will be examined by default, refer to the Date::Manip::Zones manual.

define_alias

```
$err = $tz−>define_alias($alias,$zone);
```

This will define a new alias (or override an existing alias). `$zone` must be a valid zone or an error is returned.

For more information about the different aliases which are set by default, refer to the Date::Manip::Zones manual.

If `$alias` is "reset", all aliases will be reset to the standard values. If`$zone` is ' 'reset", `$alias` will be reset to the standard value.

define_offset

```
($err,$val) = $tz−>define_offset($offset, [$dstflag,] @zone);
```

This is similar to the define_abbrev method. When an offset is encountered, all time zones which have ever included that offset are checked. This will defined which time zones, and in what order, they should be checked.

The zones to both standard and daylight saving times which include the offset (if `$dstflag` is "std" or "dst") or to only one or the other.

If `$offset` is "reset", all lists are reset to the default values. If`@zone` includes only the element 'reset', the default list and order is restored for `$offset` ($dstflag must not be given).

The following error codes are returned:

```
0  No error
1  $offset is not a valid offset in any time zone
2  $offset is not a valid offset in the selected
   time (if doing "dstonly" or "stdonly")
3  A zone (returned as $val) is not a valid time zone
4  A zone (returned as $val) does not use the offset
5  A zone (returned as $val) does not include the
   offset in the selected time (if doing "dstonly"
   or "stdonly")
   9  Offset is not a valid offset
```

periods

```
   @periods = $tz->periods($zone,$year);
```

This returns the description of all time zone periods that begin during the year given. The year is measured in universal (GMT) time.

If no time zone period starts in the given year, nothing is returned.

```
   @periods = $tz->periods($zone,undef,$year);
```

This returns all periods that begin in any year from 0001 to `$year`.

```
   @periods = $tz->periods($zone,$year0,$year1);
```

This returns all periods that begin in any year from `$year0` to `$year1`.

tzdata
tzcode

```
   $vers = $tz->tzdata();
   $vers = $tz->tzcode();
```

These return the versions of the tzdata and tzcode packages used to generate the modules.

zone

```
   $zone = $tz->zone(@args);
   @zone = $tz->zone(@args);
```

This function will return a list of all zones, or the default zone, which matches all of the supplied information. In scalar context, it will return only the default zone. In list context, it will return all zones.

`@args` may include any of the following items, and the order is not important.

```
   A zone name or alias ($alias)

   A zone abbreviation ($abbrev)

   An offset ($offset)

   A dstflag ($dstflag)

   A date ($date)
```

It is NOT valid to include two of any of the items. Any time zone returned will match all of the data supplied.

If an error occurs, undef is returned. If no zone matches, an empty string, or an empty list is returned.

The order of the zones will be determined in the following way:

If `$abbrev` is given, the order of time zones will be determined by it (and `$dstflag`). If `$dstflag` is "std", all zones which match `$abbrev` in standard time are included, followed by all that match `$abbrev` in saving time (but no duplication is allowed). The reverse is true if `$dstflag` is "dst".

If `$abbrev` is not given, but `$offset` is, `$offset` (and `$dstflag`) will determine the order given. If `$dstflag` is "std", all zones which match `$offset` in standard time are included, followed by all that match `$offset` in saving time (but no duplication is allowed). The reverse is true if `$dstflag` is "dst".

If `$date` is given, only zones in which `$date` will appear in a zone that matches all other information are given. `$date` is a wall clock time.

If no `$zone`, `$abbrev`, or `$offset` are entered, the local time zone may be returned (unless `$date` is entered, and it doesn't exist in the local time zone).

NOTE: there is one important thing to note with respect to `$dstflag` when you are working with a timezone expressed as an offset and a date is passed in. In this case, the default value of `$dstflag` is ‘‘dst’’ (NOT ‘‘stdonly’’), and you probably never want to pass in a value of ‘‘std’’ (though passing in ‘‘stdonly’’ is okay).

For standard offsets (with no minute component), there is always a standard timezone which matches that offset. For example, the timezone ‘‘+0100’’ matches the timezone ‘‘Etc/GMT+01’’, so you will never get a timezone in daylight saving time if `$dstflag` is ‘‘std’’.

If you want to pass in a date of 2001−07−01−00:00:00 and an timezone of ‘‘+0100’’ and you want to get a timezone that refers to that date as a daylight saving time date, you must use the `$dstflag` of ‘‘dst’’ (or ‘‘dstonly’’).

Because this is almost always the behavior desired, when a zone is passed in as an offset, and a date is passed in, the default `$dstflag` is ‘‘dst’’ instead of ‘‘std’’. In all other situations, the default is still ‘‘std’’.

If the timezone is expressed as an abbreviation, this problem does not occur.

TIME ZONE INFORMATION IN DATE::MANIP

Date::Manip makes use of three potentially different time zones when working with a date.

The first time zone that may be used is the actual local time zone. This is the time zone that the computer is actually running in.

The second time zone is the working time zone. Usually, you will want the default time zone to be the local time zone, but occasionally, you may want the default time zone to be different.

The third time zone is the actual time zone that was parsed, or set, for a date. If a date contains no time zone information, it will default to the working time zone.

The local time zone is determined using the methods described in the following section. Methods exist for locating the zone in one of the system configuration files, determining it by running a system command, or by looking it up in the registry (for Windows operating systems). If all of these methods fail, the local time zone may be set using either the `$::TZ` or `$ENV`{TZ} variables. Please note that these should ONLY be used to set the actual local time zone.

If you are running in one time zone, but you want to force dates to be specified in an alternate time zone by default, you need to set the working time zone. The working time zone defaults to the local time zone, but this can be changed using either the SetDate or ForceDate config variables. Refer to the Date::Manip::Config manual for more information.

Finally, when a date is actually parsed, if it contains any time zone information, the date is stored in that time zone.

DETERMINING THE SYSTEM TIME ZONE

There are a large number of ways available for determining the time zone. Some or all of them may be checked. A list of methods to use is provided by default, and may be overridden by the curr_zone_methods function described above. To override the default order and/or list of methods, just pass in a list of method names (with arguments where necessary), and only those methods will be done, and in the order given.

The following methods are available:

```
Method      Argument(s)     Procedure
======      ===========     =========

main        VAR             The main variable named VAR is
                            checked. E.g. "main TZ" checks
                            the variable $::TZ .
```

```
env        TYPE VAR       The named environment variable
                          is checked and the type of
                          data stored there (TYPE can
                          be 'zone' or 'offset' which
                          is the number of seconds from
                          UTC).

file       FILE           Look in the given file for any
                          one of the following case
                          insensitive lines:
                             ZONE
                             tz = ZONE
                             zone = ZONE
                             timezone = ZONE
                          ZONE may be quoted (single or
                          double) and whitespace is
                          ignored (except that underscores
                          in the zone name may be replaced
                          by whitespace on some OSes). If
                          the entire line is a zone, it must
                          be the first non-blank non-comment
                          line in the file.

command    COMMAND        Runs a command which produces
                          a time zone as the output.

cmdfield   COMMAND N      Runs a command which produces
                          whitespace separated fields,
                          the Nth one containing the
                          time zone (fields are numbered
                          starting at 0, or from the
                          end starting at -1).

gmtoff                    Uses the current offset from
                          GMT to come up with a best guess.

tzdata     FILE DIR       This uses a system config file that
                          contains a pointer to the local tzdata
                          files to  determine the timezone.  On
                          many operating systems, use:

                          tzdata /etc/localtime /usr/share/zoneinfo

                          FILE is the system file.  DIR is the
                          directory where the tzdata files are stored.

                          The config file is either a link to a file
                          in the tzdata directory or a copy of one
                          of the files.

registry                  Look up the value in the
                          Windows registry. This is only
                          available to hosts running a
                          Windows operating system.
```

Note that the ‘‘main’’ and ‘‘env’’ methods should only be used to specify the actual time zone the system is running in. Use the SetDate and ForceDate config variables to specify an alternate time zone that you want to work in.

By default, the following methods are checked (in the order given) on Unix systems:

```
main     TZ
env      zone TZ
file     /etc/TIMEZONE
file     /etc/timezone
file     /etc/sysconfig/clock
file     /etc/default/init
command  "/bin/date +%Z"
command  "/usr/bin/date +%Z"
command  "/usr/local/bin/date +%Z"
cmdfield /bin/date            −2
cmdfield /usr/bin/date        −2
cmdfield /usr/local/bin/date  −2
command  "/bin/date +%z"
command  "/usr/bin/date +%z"
command  "/usr/local/bin/date +%z"
tzdata   /etc/localtime /usr/share/zoneinfo
gmtoff
```

The default methods for Windows systems are:

```
main     TZ
env      zone TZ
registry
gmtoff
```

The default methods for VMS systems are:

```
main     TZ
env      zone TZ
env      zone SYS$TIMEZONE_NAME
env      zone UCX$TZ
env      zone TCPIP$TZ
env      zone MULTINET_TIMEZONE
env      offset SYS$TIMEZONE_DIFFERENTIAL
gmtoff
```

The default methods for all other systems are:

```
main     TZ
env      zone TZ
gmtoff
```

If anyone wants better support for a specific OS, please contact me and we’ll coordinate adding it.

In all cases, the value returned from the method may be any of the following:

```
the full name of a time zone (e.g. America/New_York)
or an alias

an abbreviation (e.g. EDT) which will be used to
determine the zone if possible

an offset (+hh, +hhmn, +hh:mm, +hh:mm:ss) from GMT
```

The Date::Manip::Zones module contains information about the time zones and aliases available, and what time zones contain the abbreviations.

DESIGN ISSUES

The design decisions made in writing this module may cause some questions (and probably complaints). The time zone modules are all generated using scripts (included in the Date::Manip distribution) which use the standard tzdata tools to parse the tzdata files and store that information in perl modules.

I'd like to address some of them, to avoid answering some of the ''why did you do it that way'' remarks. I do welcome discussion about these decisions... but preferably after you understand why those decisions were made so that that we have an informed basis to begin a discussion.

Why not use existing zoneinfo files

Some people will probably think that I should have written an interface to the zoneinfo files which are distributed with most operating systems. Although I considered doing that, I rejected the idea for two reasons.

First, not all operating systems come with the zoneinfo databases in a user accessible state (Microsoft for example). Even those that do include them store the information in various formats and locations. In order to bypass all that, I have included the data directly in these modules.

Second, as I was doing my initial investigations into this, I ran into a bug in the Solaris zoneinfo tools (long since fixed I'm sure). I decided then that I didn't want to depend on an implementation where I could not control and fix the bugs.

Why not use the native tzdata files

Another decision people may question is that I parse the tzdata files and store the data from them in a large number of perl modules instead of creating an interface to the tzdata files directly. This was done solely for the sake of speed. Date::Manip is already a slow module. I didn't want to slow it down further by doing the complex parsing required to interpret the tzdata files while manipulating dates. By storing the data in these modules, there is little or no parsing done while using Date::Manip modules. It costs a little disk space to store this information... but very little of it is actually loaded at runtime (time zone data is only loaded when the time zone is actually referred to), so I feel it's a good tradeoff.

Why store the information in so many files

The data from the native tzdata files are parsed and stored in two sets of modules. These include almost 500 Date::Manip::Offset::* modules and almost 450 Date::Manip::TZ::* modules.

I note that on my linux box, /usr/share/zoneinfo (which contains data files generated from the tzdata files) contains over 1700 files, so I'm not doing anything ''new'' by breaking up the information into separate files. And doing so has a huge impact on performance... it is not necessary to load and/or manipulate data from time zones which are not in use.

The minute I made the decision to distribute the timezone information myself, as opposed to using the system version, it was a given that there would be a lot of files.

These modules are loaded only when the time zone or offset is actually used, so, unless dates from around the world are being parsed, only a very small number of these modules will actually be loaded. In many applications, only a single TZ module will be loaded. If parsing dates which have timezone information stored as offsets, one or two Offset modules will also be loaded.

The disk space seems excessive

Currently, the disk usage of the perl files is around 9 MB. Total disk usage for /usr/share/zoneinfo on my computer is around 4 MB. There are a couple of differences.

The primary difference is that the zoneinfo files are stored in a binary (and hence, more compressed) version, where the perl modules have all the data in pure text.

Since these are all automatically generated and used, it may be beneficial to store the data in some packed binary format instead of the fully expanded text form that is currently in use. This would decrease the disk space usage, and might improve performance. However, the performance improvement would happen only once per timezone, and would make for more complicated code, so I'm not very interested in pursuing this.

Another aspect of the current modules is that they all include pod documentation. Although not necessary, this allows users to easily see what modules handle which time zones, and that's nice. It also allows me to use pod_coverage tests for the module which is a nice check to make sure that the documentation is accurate.

All told, I don't consider the disk usage excessive at all.

KNOWN PROBLEMS OR ISSUES

Unable to determine Time Zone

When using Date::Manip, when the module is initialized, it must be able to determine the local time zone. If it fails to do so, an error will occur:

```
   Unable to determine Time Zone
```

and the script will exit.

In the past, this was the most common problem with using Date::Manip . With the release of 6.00, this problem should be significantly less common. If you do get this error, please refer to the section above DETERMINING THE SYSTEM TIME ZONE for information about determining the local time zone. I am also interested in hearing about this so that I can update the default list of methods to be able to determine the local time zone better.

Asia/Jerusalem time zone

The Asia/Jerusalem time zone has a non-standard way of specifying the start and end of Daylight Saving Time based on the Hebrew calendar.

As a result, there is no way to specify a simple rule to define time zone changes for all years in the future. As such, this module supports all time zone changes currently specified in the zoneinfo database (which currently goes to the year 2037) but does not attempt to correctly handle zone changes beyond that date. As a result, Date::Manip should not be used to parse dates in the Jerusalem time zone that are far enough in the future that information is not included in the current version of the zoneinfo database.

LMT and zzz abbreviations

Both the LMT and zzz abbreviations are used in the zoneinfo databases. LMT is use for most time zones for the times before the Gregorian calendar was adopted, and zzz is used for a few where the time zone was created and no description of dates prior to that are supported. Both LMT and zzz are basically ignored in parsing dates (because there is no reasonable way to determine which zone they are referring to), and will be treated as the local time zone regardless.

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)