NAME

Date::Manip::Objects − A description of the various Date::Manip objects

SYNOPSIS

The Date::Manip package consist of several modules, each of which perform a set of operations on a specific class of objects. This document describes how the various modules work together.

DESCRIPTION

Date::Manip consists of the following primary modules:

Date::Manip::Obj

The Date::Manip::Obj module is not intended for direct use. It is used as a base class for all other Date::Manip classes described below.

The Date::Manip::Obj module contains some functions which are inherited by all these classes, so to understand all of the methods available to any of the classes below, you must include those documented in the Date::Manip::Obj class.

Date::Manip::Base

The Date::Manip::Base is used to perform basic operations including basic date operations, management of configuration options, handling the definitions used in different languages, etc.

A Date::Manip::Base object does not, of itself, contain any date information. Instead, it contains configuration information which determines how the Date::Manip package performs date operations. The configuration information is documented in the Date::Manip::Config document.

The Date::Manip::Base object has one other property that is very important. When performing basic date operations, some intermediate results are cached in the object which leads to significant performance increases in later operations. As such, it is important to reuse the object as much as possible, rather than creating new Date::Manip::Base objects all the time.

Much of the information in this document is related to this issue, and tells how to create various higher-level objects in order to get the most efficient reuse of this cached data.

Because all other objects depend on a Date::Manip::Base object, a Date::Manip::Base object is embedded in all other objects, and the same Base object can be shared by any number of objects to achieve maximum performance.

Date::Manip::TZ

The Date::Manip::TZ module adds support for time zones. It is used to verify date and time zone information, convert dates from one time zone to another, and handle all daylight saving time transitions.

Similar to the Date::Manip::Base object, a great deal of information is cached in the Date::Manip::TZ object. This includes lists of all time zones, offsets, and abbreviations for all time zones. It also includes more a more detailed description of every time zone that has actually been worked used.

A Date::Manip::TZ object relies on a Date::Manip::Base object (and a Date::Manip::Base object is always embedded in a Date::Manip::TZ object). All higher level objects (those listed next) depend on both a Date::Manip::Base and Date::Manip::TZ object, so a Date::Manip::TZ object is embedded in them.

In order to achieve maximum performance, and minimize memory usage, a Date::Manip::TZ object can be shared by any number of higher level objects, and in fact, it is desirable to reuse the same Date::Manip::TZ object as often as possible.

Date::Manip::Date
Date::Manip::Delta
Date::Manip::Recur

These are the primary modules which are used to perform all high level date operations.

The Date::Manip::Date class performs operations on dates (which includes a date, time, and time zone). The Date::Manip::Delta class performs operations with deltas (amounts of time). The

Date::Manip::Recur class performs operations on recurring events.

As mentioned above, each of these high level classes rely on both a Date::Manip::TZ object and a Date::Manip::Base object, so a Date::Manip::TZ object is embedded in each one (and the Date::Manip::TZ object has a Date::Manip::Base object embedded in it).

A Date::Manip::Date object contains a single date, so in order to work with multiple dates, multiple Date::Manip::Date objects will need to be created. In order to make the most effective use of cached information in the Date::Manip::Base object, the same Date::Manip::TZ object can be embedded in each of the higher level objects.

The same goes for multiple Date::Manip::Delta and Date::Manip::Recur objects.

There are also many secondary modules including:

```
Date::Manip::TZ_Base
Date::Manip::TZdata
Date::Manip::Zones
Date::Manip::Lang::*
Date::Manip::TZ::*
Date::Manip::Offset::*
```

None of these are intended to be used directly.

WORKING WITH DATE::MANIP OBJECTS (SINGLE CONFIGURATION)

By far the most common usage of Date::Manip involves setting a single local time zone, parsing dates in a single language, and having all other configuration parameters set to a single value that doesn't change over the course of the program.

Whenever this is the case, you can use the methods listed in this section to create any number of Date::Manip objects. It will automatically optimize the use of cached data to get the best performance.

If you do need to work with multiple different configurations (such as parsing dates from multiple languages), please refer to the next section ''WORKING WITH DATE::MANIP OBJECTS (MULTIPLE CONFIGURATION)''.

Working with high level objects

The most common situation is one where you will need to use one or more high level objects (Date, Delta, or Recur objects). In addition, you may want to use the lower level (Base or TZ) objects.

The first thing you should do is to create your initial object. Create the highest level object you will be using. For example if you will be working with dates, create the first date object with:

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

The next step is to set the configuration values. Use the config method to do this:

```
$date->config(ARGS);
```

Although you can call the config method later, it is strongly suggested that the configuration be set soon after the initial object is created and not altered later. Every time you alter the configuration, some of the cached data is cleared, so for optimal performance, you don't want to alter the configuration if possible.

Additional high-level objects can be created using the calls:

```
$date2 = $date->new_date();
$delta = $date->new_delta();
$recur = $date->new_recur();
```

To access the embedded Date::Manip::TZ and Date::Manip::Base objects, use the calls:

```
$tz    = $date->tz();
$base  = $date->base();
```

Working with low level objects only

If you will only be working with low level objects, create them with one of the calls:

```
$tz    = new Date::Manip::TZ;
$base  = new Date::Manip::Base;
```

To get the base object embedded in a Date::Manip::TZ object, use:

```
$base  = $tz->base();
```

For a more complete description of the methods used here, refer to the Date::Manip::Obj document.

WORKING WITH DATE::MANIP OBJECTS (MULTIPLE CONFIGURATION)

Occasionally, it may be useful to have multiple sets of configurations. In order to do this, multiple Date::Manip::Base objects must be created (each with their own set of configuration options), and then new Date::Manip objects are created with the appropriate Date::Manip::Base object embedded in them.

Possible reasons include:

Parsing multiple languages

A Date::Manip::Base object includes information about a single language. If you need to parse dates from two (or more) languages, a Date::Manip::Base object needs to be created for each one. This could be done as:

```
$date_eng1 = new Date::Manip::Date;
$date_eng1->config("language","English");

$date_spa1 = new Date::Manip::Date;
$date_spa1->config("language","Spanish");
```

Any additional Date::Manip objects created from the first will work with English. Additional objects created from the second will work in Spanish.

Business modes for different countries and/or businesses

If you are doing business mode calculations (see Date::Manip::Calc) for two different businesses which have different holiday lists, work weeks, or business days, you can create different objects which read different config files (see Date::Manip::Config) with the appropriate description of each.

The primary issue when dealing with multiple configurations is that it is necessary for the programmer to manually keep track of which Date::Manip objects work with each configuration. For example, refer to the following lines:

```
$date1 = new Date::Manip::Date [$opt1,$val1];
$date2 = new Date::Manip::Date $date1, [$opt2,$val2];
$date3 = new Date::Manip::Date $date1;
$date4 = new Date::Manip::Date $date2;
```

The first line creates 3 objects: a Date::Manip::Base object, a Date::Manip::TZ object, and a Date::Manip::Date object). The Date::Manip::Base object has the configuration set to contain the value(s) passed in as the final list reference argument.

The second line creates 3 new objects (a second Date::Manip::Base object, a second Date::Manip::TZ object, and a second Date::Manip::Date object). Since a list reference containing config variables is passed in, a new Date::Manip::Base object is created, rather than reusing the first one. The second Date::Manip::Base object contains all the config from the first, as well as the config variables passed in in the list reference argument.

The third line creates another Date::Manip::Date object which uses the first Date::Manip::Base and Date::Manip::TZ objects embedded in it.

The fourth line creates another Date::Manip::Date object which uses the second Date::Manip::Base and Date::Manip::TZ objects embedded in it.

Most of the time there will only be one set of configuration options used, so this complexity is really for a

very special, and not widely used, bit of functionality.

WORKING WITH DATE::MANIP OBJECTS (ADDITIONAL NOTES)

object reuse

In order to create additional Date::Manip objects, a previously created object should be passed in as the first argument. This will allow the same Base object to be embedded in both in order to maximize data reuse of the cached intermediate results, and will result in much better performance. For example:

```
$date1 = new Date::Manip::Date;
$date2 = new Date::Manip::Date $date1;
```

This is important for two reasons. First is memory usage. The Date::Manip::Base object is quite large. It stores a large number of precompile regular expressions for language parsing, and as date operations are done, intermediate results are cached which can be reused later to improve performance. The Date::Manip::TZ object is even larger and contains information about all known time zones indexed several different ways (by offset, by abbreviation, etc.). As time zones are actually used, a description of all of the time change rules are loaded and added to this object.

Since these objects are so large, it is important to reuse them, rather than to create lots of copies of them. It should be noted that because these objects are embedded in each of the high level object (Date::Manip::Date for example), it makes these objects appear quite large.

The second reason to reuse Date::Manip::Base objects is performance. Since intermediate results are cached there, many date operations only need to be done once and then they can be reused any number of times. In essence, this is doing the same function as the Memoize module, but in a more efficient manner. Memoize caches results for function calls. For Date::Manip, this would often work, but if you change a config variable, the return value may change, so Memoize could cause things to break. In addition, Memoize caches primarily at the function level, but Date::Manip stores caches intermediate results wherever performance increase is seen. Every time I consider caching a result, I run a test to see if it increases performance. If it doesn't, or it doesn't make a significant impact, I don't cache it.

Because the caching is quite finely tuned, it's much more efficient than using a generic (though useful) tool such as Memoize.

configuration changes

As a general rule, you should only pass in configuration options when the first object is created. In other words, the following behavior is discouraged:

```
$date = new Date::Manip::Date;
$date->config(@opts);

... do some stuff

$date->config(@opts);

... do some other stuff
```

Because some of the cached results are configuration specific, when a configuration change is made, some of the cached data must be discarded necessitating those results to be recalculated.

If you really need to change configuration in the middle of execution, it is certainly allowed of course, but if you can define the configuration once immediately after the object is first created, and then leave the configuration alone, performance will be optimized.

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)