NAME

Log::Message - A generic message storing mechanism;

SYNOPSIS

```
use Log::Message private => 0, config => '/our/cf_file';

my $log = Log::Message->new(    private => 1,
                                level   => 'log',
                                config  => '/my/cf_file',
                            );

$log->store('this is my first message');

$log->store(    message => 'message #2',
                tag     => 'MY_TAG',
                level   => 'carp',
                extra   => ['this is an argument to the handler'],
           );

my @last_five_items = $log->retrieve(5);

my @items = $log->retrieve( tag     => qr/my_tag/i,
                            message => qr/\d/,
                            remove  => 1,
                          );

my @items = $log->final( level => qr/carp/, amount => 2 );

my $first_error = $log->first()

# croak with the last error on the stack
$log->final->croak;

# empty the stack
$log->flush();
```

DESCRIPTION

Log::Message is a generic message storage mechanism. It allows you to store messages on a stack -- either shared or private -- and assign meta-data to it. Some meta-data will automatically be added for you, like a timestamp and a stack trace, but some can be filled in by the user, like a tag by which to identify it or group it, and a level at which to handle the message (for example, log it, or die with it)

Log::Message also provides a powerful way of searching through items by regexes on messages, tags and level.

Hierarchy

There are 4 modules of interest when dealing with the Log::Message::* modules:

Log::Message

> Log::Message provides a few methods to manipulate the stack it keeps. It has the option of keeping either a private or a public stack. More on this below.

Log::Message::Item

These are individual message items, which are objects that contain the user message as well as the meta-data described above. See the *Log::Message::Item* manpage to see how to extract this meta-data and how to work with the Item objects. You should never need to create your own Item objects, but knowing about their methods and accessors is important if you want to write your own handlers. (See below)

Log::Message::Handlers

These are a collection of handlers that will be called for a level that is used on a *Log::Message::Item* object. For example, if a message is logged with the 'carp' level, the 'carp' handler from *Log::Message::Handlers* will be called. See the *Log::Message::Handlers* manpage for more explanation about how handlers work, which one are available and how to create your own.

Log::Message::Config

Per Log::Message object, there is a configuration required that will fill in defaults if the user did not specify arguments to override them (like for example what tag will be set if none was provided), *Log::Message::Config* handles the creation of these configurations.

Configuration can be specified in 4 ways:

- As a configuration file when you `use Log::Message`
- As arguments when you `use Log::Message`
- As a configuration file when you create a new *Log::Message* object. (The config will then only apply to that object if you marked it as private)
- As arguments when you create a new Log::Message object.

  You should never need to use the *Log::Message::Config* module yourself, as this is transparently done by *Log::Message*, but its manpage does provide an explanation of how you can create a config file.

Options

When using Log::Message, or creating a new Log::Message object, you can supply various options to alter its behaviour. Of course, there are sensible defaults should you choose to omit these options.

Below an explanation of all the options and how they work.

config

The path to a configuration file to be read. See the manpage of *Log::Message::Config* for the required format

These options will be overridden by any explicit arguments passed.

private

Whether to create, by default, private or shared objects. If you choose to create shared objects, all Log::Message objects will use the same stack.

This means that even though every module may make its own $log object they will still be sharing the same error stack on which they are putting errors and from which they are retrieving.

This can be useful in big projects.

If you choose to create a private object, then the stack will of course be private to this object, but it will still fall back to the shared config should no private config or overriding arguments be provided.

verbose

Log::Message makes use of another module to validate its arguments, which is called *Params::Check*, which is a lightweight, yet powerful input checker and parser. (See the

Params::Check manpage for details).

The verbose setting will control whether this module will generate warnings if something improper is passed as input, or merely silently returns undef, at which point Log::Message will generate a warning.

It's best to just leave this at its default value, which is '1'

tag

The tag to add to messages if none was provided. If neither your config, nor any specific arguments supply a tag, then Log::Message will set it to 'NONE'

Tags are useful for searching on or grouping by. For example, you could tag all the messages you want to go to the user as 'USER ERROR' and all those that are only debug information with 'DEBUG'.

At the end of your program, you could then print all the ones tagged 'USER ERROR' to STDOUT, and those marked 'DEBUG' to a log file.

level

`level` describes what action to take when a message is logged. Just like `tag`, Log::Message will provide a default (which is 'log') if neither your config file, nor any explicit arguments are given to override it.

See the Log::Message::Handlers manpage to see what handlers are available by default and what they do, as well as to how to add your own handlers.

remove

This indicates whether or not to automatically remove the messages from the stack when you've retrieved them. The default setting provided by Log::Message is '0': do not remove.

chrono

This indicates whether messages should always be fetched in chronological order or not. This simply means that you can choose whether, when retrieving items, the item most recently added should be returned first, or the one that had been added most long ago.

The default is to return the newest ones first

Methods

new

This creates a new Log::Message object; The parameters it takes are described in the `Options` section below and let it just be repeated that you can use these options like this:

```
my $log = Log::Message->new( %options );
```

as well as during `use` time, like this:

```
use Log::Message option1 => value, option2 => value
```

There are but 3 rules to keep in mind:

- Provided arguments take precedence over a configuration file.
- Arguments to new take precedence over options provided at `use` time
- An object marked private will always have an empty stack to begin with

store

This will create a new Item object and store it on the stack.

Possible arguments you can give to it are:

message

This is the only argument that is required. If no other arguments are given, you may even leave off the `message` key. The argument will then automatically be assumed to be the message.

tag

The tag to add to this message. If not provided, Log::Message will look in your configuration for one.

level

The level at which this message should be handled. If not provided, Log::Message will look in your configuration for one.

extra

This is an array ref with arguments passed to the handler for this message, when it is called from store();

The handler will receive them as a normal list

store() will return true upon success and undef upon failure, as well as issue a warning as to why it failed.

retrieve

This will retrieve all message items matching the criteria specified from the stack.

Here are the criteria you can discriminate on:

tag

A regex to which the tag must adhere. For example `qr/\w/`.

level

A regex to which the level must adhere.

message

A regex to which the message must adhere.

amount

Maximum amount of errors to return

chrono

Return in chronological order, or not?

remove

Remove items from the stack upon retrieval?

In scalar context it will return the first item matching your criteria and in list context, it will return all of them.

If an error occurs while retrieving, a warning will be issued and undef will be returned.

first

This is a shortcut for retrieving the first item(s) stored on the stack. It will default to only retrieving one if called with no arguments, and will always return results in chronological order.

If you only supply one argument, it is assumed to be the amount you wish returned.

Furthermore, it can take the same arguments as `retrieve` can.

last

This is a shortcut for retrieving the last item(s) stored on the stack. It will default to only retrieving one if called with no arguments, and will always return results in reverse chronological order.

If you only supply one argument, it is assumed to be the amount you wish returned.

Furthermore, it can take the same arguments as `retrieve` can.

flush

This removes all items from the stack and returns them to the caller

SEE ALSO

Log::Message::Item, *Log::Message::Handlers*, *Log::Message::Config*

AUTHOR

This module by Jos Boumans <kane@cpan.org>.

Acknowledgements

Thanks to Ann Barcomb for her suggestions.

COPYRIGHT