3. Custom formatting

Gabi Melman edited this page on Apr 11 · 94 revisions

Each logger's sink have a formatter which formats the messages to its destination.

spdlog's default logging format is in the form of:

```
[2014-10-31 23:46:59.678] [my_loggername] [info] Some message
```

There are 2 ways to customize a logger's format:

- Set the pattern string (*recommended*):

```
set_pattern(pattern_string);
```

- Or implement custom formatter that implements the formatter interface and call

```
set_formatter(std::make_unique<my_custom_formatter>());
```

Customizing format using `set_pattern(..)`

Format can be applied globally to **all registered** loggers:

```
spdlog::set_pattern("*** [%H:%M:%S %z] [thread %t] %v ***");
```

or to a specific **logger** object:

```
some_logger->set_pattern(">>>>>>>>> %H:%M:%S %z %v <<<<<<<<<");
```

or to a specific **sink** object:

```
some_logger->sinks()[0]->set_pattern(">>>>>>>>> %H:%M:%S %z %v <<<<<<<<<");
some_logger->sinks()[1]->set_pattern("..");
```

Efficiency

Whenever the user calls `set_pattern("..")` the library "compiles" the new pattern to an internal efficient representation - This way the performance stays excellent even with complex patterns (no re-parsing of the pattern on each log call).

Pattern flags

Pattern flags are in the form of `%flag` and resembles the strftime function:

flag	meaning	example
%v	The actual text to log	"some user text"
%t	Thread id	"1232"
%P	Process id	"3456"
%n	Logger's name	"some logger name"
%l	The log level of the message	"debug", "info", etc
%L	Short log level of the message	"D", "I", etc
%a	Abbreviated weekday name	"Thu"
%A	Full weekday name	"Thursday"
%b	Abbreviated month name	"Aug"
%B	Full month name	"August"
%c	Date and time representation	"Thu Aug 23 15:35:46 2014"
%C	Year in 2 digits	"14"
%Y	Year in 4 digits	"2014"
%D or %x	Short MM/DD/YY date	"08/23/14"
%m	Month 01-12	"11"
%d	Day of month 01-31	"29"
%H	Hours in 24 format 00-23	"23"
%I	Hours in 12 format 01-12	"11"

flag	meaning	example
%M	Minutes 00-59	"59"
%S	Seconds 00-59	"58"
%e	Millisecond part of the current second 000-999	"678"
%f	Microsecond part of the current second 000000-999999	"056789"
%F	Nanosecond part of the current second 000000000-999999999	"256789123"
%p	AM/PM	"AM"
%r	12 hour clock	"02:55:02 pm"
%R	24-hour HH:MM time, equivalent to %H:%M	"23:55"
%T or %X	ISO 8601 time format (HH:MM:SS), equivalent to %H:%M:%S	"23:55:59"
%z	ISO 8601 offset from UTC in timezone ([+/-]HH:MM)	"+02:00"
%E	Seconds since the epoch	"1528834770"
%%	The % sign	"%"
%+	spdlog's default format	"[2014-10-31 23:46:59.678] [mylogger] [info] Some message"
%^	start color range (can be used only once)	"[mylogger] [info(green)] Some message"
%$	end color range (for example %^[+++]%$ %v) (can be used only once)	[+++] Some message
%@	Source file and line (use SPDLOG_TRACE(..), SPDLOG_INFO(...) etc. instead of spdlog::trace(...)	my_file.cpp:123
%s	Basename of the source file (use SPDLOG_TRACE(..), SPDLOG_INFO(...) etc.)	my_file.cpp
%g	Full or relative path of the source file as appears in the __FILE__ macro (use SPDLOG_TRACE(..), SPDLOG_INFO(...) etc.)	/some/dir/my_file.cpp

flag	meaning	example
%#	Source line (use SPDLOG_TRACE(..), SPDLOG_INFO(...) etc.)	123
%!	Source function (use SPDLOG_TRACE(..), SPDLOG_INFO(...) etc. see tweakme for pretty-print)	my_func
%o	Elapsed time in milliseconds since previous message	456
%i	Elapsed time in microseconds since previous message	456
%u	Elapsed time in nanoseconds since previous message	11456
%0	Elapsed time in seconds since previous message	4

Aligning

Each pattern flag can can be aligned by prepending a width number(upto 64).

Use `-` (left align) or `=` (center align) to control the align side:

align	meaning	example	result
%<width><flag>	Right align	%8l	" info"
%-<width><flag>	Left align	%-8l	"info "
%=<width><flag>	Center align	%=8l	" info "

Optionally add `!` to truncate the result if its size exceeds the specified width:

align	meaning	example	result
%<width>!<flag>	Right align or truncate	%3!l	"inf"
-%<width>!<flag>	Left align or truncate	%-2!l	"in"
=%<width>!<flag>	Center align or truncate	%=1!l	"i"

Note: To truncate function names use '!!'. For example `%10!!` will limit function names to 10 chars.

Extending spdlog with your own flags

You can define your own flags by inheriting the custom_flag_formatter class and implementing the `clone()` and `format(...)` abstract methods.

The following example adds a new flag **%*** - which will be bound to a <my_formatter_flag> instance.

```
#include "spdlog/pattern_formatter.h"
class my_formatter_flag : public spdlog::custom_flag_formatter
{
public:
    void format(const spdlog::details::log_msg &, const std::tm &, spdlog::memory_
    {
        std::string some_txt = "custom-flag";
        dest.append(some_txt.data(), some_txt.data() + some_txt.size());
    }

    std::unique_ptr<custom_flag_formatter> clone() const override
    {
        return spdlog::details::make_unique<my_formatter_flag>();
    }
};

void custom_flags_example()
{
    auto formatter = std::make_unique<spdlog::pattern_formatter>();
    formatter->add_flag<my_formatter_flag>('*').set_pattern("[%n] [%*] [%^%l%$] %v
    spdlog::set_formatter(std::move(formatter));
}
```

Note: The `clone()` method should return a deep copy of the object and is required because spdlog passes a new copy of the object to each sink in use (for performance reasons as this enables to have a state in this object without worrying about race conditions or thread safety across sinks).

Note: You can also override spdlog's built in flags in this way.