Date-time, C functions for date and time handling.

Manual for Release 1v1 25/5/2022

# Introduction

These files provide C and C++ functions for date handling.

The dates/times are internally stored in a struct tm and a struct strp\_tz\_struct, and can be converted into seconds since 1st January 1970 by using ya\_mktime().

See time\_local.h for the full list of functions defined here and a brief summary of their use.

The file main.c contains a complete test program with a large number of examples.

As an example

char \* ya\_strptime(const char \*s, const char \*format, struct tm \*tp);

converts the input string s which is assumed to contain a date & time described by format into data stored into tp.

time\_t ya\_mktime(struct tm \*tp);

then converts the resultant structure (tp) into seconds since 1st January 1970. Note that time\_local.h defines time\_t as a 64 bit signed integer (even when compiling for a 32 bit target). Also note that ya\_mktime() can also change the elements of tp – potentially filling in missing elements (e.g. the day of the week) and normalising others.

If fractions of a second are required then they are available in the f\_secs element of the strp\_tz structure. This means that these functions can provide sub femtosecond resolution over a range of just over +/- two billion years.

For example a format of “%d-%b-%y %H:%M:%S.%f” is designed to read date/times like

05-Jul-19 11:59:24.037063

Another example of a format is “%Y-%m-%d %H:%M:%S.%f” which will read dates/times like:

2019-12-31 00:00:01.567000

Also,

size\_t ya\_strftime(char \*s, size\_t maxsize, const char \*format, const struct tm \*tp);

does the opposite of ya\_strptime(), in that it will convert to values in tp to a string s of maximum length maxsize based on the specified format.

Care has been taken to make the process “round loop” exact so converting a string with ya\_strptime() can be reversed exactly by passing the resultant tp to ya\_strftime().

Equally

void sec\_to\_tm(time\_t t,struct tm \*tp);

is the reverse of ya\_mktime() in that it converts seconds after the epoch to a struct tm.

The full list of format specifiers is given in Appendix A and Appendix B, but the ones most commonly used are:

* %b Month in text (either as 3 characters like *Jul* or in full like *December*)
* %d Day of the month as a one or 2 digit integer number (e.g., 05 in the first example)
* %f Fractional part of the seconds (e.g., 037063 in the first example)
* %H Hours (one or 2 digit integer number) (e.g., 11 in the first example)
* %m Month as a one or 2 digit integer number (e.g., 12 in the second example)
* %M Minutes as a 1 or 2 digit integer number (e.g., 59 in the first example)
* %S Seconds as a 1 or 2 digit integer number (e.g., 24 in the first example)
* %y Year as a 2 digit integer number (e.g., 19 in the first example)
* %Y Year as a 4 digit integer number (e.g., 2019 in the second example)
* %% matches a percent sign (%)
* Space matches any “whitespace” in the input (e.g., the gap between the date and the time in the examples).
* Any other character is matched exactly (e.g., the two –‘s, two :’s and a . in the examples). This can also be used to match double quotes (“) if these surround the date/time.

# Installation

Download the files from github, then compile the test program as shown below.

These files compile and run correctly with devC++/TDMgcc 10.3.0 on Windows, Builder C++ (10.2) on Windows and gcc 9.4.0 and 10.3.0 on Ubuntu Linux.

For gcc under Linux compile the test program with:

gcc -Wall -O3 -o date-time main.c strftime.c strptime.c

./date-time

For dev-C++ (tested with tdmgcc 10.3.0) there is a \*.dev file, and the project files (\*.cbproj) for Builder C++ (10.2) are also present.

You should see no errors or warnings when compiling these files.

In all cases when running the executable, you should see lots of output with the last line reading:

7300275 tests conducted; no errors found

See the file LICENSE for details, but date-time is free under the MIT license for both commercial and non-commercial use.

To use these within another program (C or C++) insert the following at the start of the file:

#include <time.h>

#include <stdbool.h>

#include <stdint.h>

#include <limits.h>

#include "time\_local.h"

/\* the two #defines below sets this code to use the new functions defined in strptime.c & strftime.c rather than the ones (that might be) in the standard libraries \*/

// char \* ya\_strptime(const char \*s, const char \*format, struct tm \*tm);

#define strptime(s,f,m) ya\_strptime(s,f,m)

// size\_t ya\_strftime(char \*s, size\_t maxsize, const char \*format, const struct tm \*timeptr);

#define strftime(s,m,f,t) ya\_strftime(s,m,f,t)

# Calendar and assumptions

These functions assume the Gregorian calendar, which was introduced in 1582 and its use gradually expanded across the world, being used in Great Britain and the USA from 1752, Greece since 1923, etc.

In more detail, they use a proleptic Gregorian calendar (i.e., it’s expanded backwards before its introduction as if it has always been used) and use astronomical year numbering (so there is a year 0). This is the approach taken by ISO 8601. The advantage of this approach is that there are no gaps or jumps in the calendar (so for example the day before Tuesday is always Monday and the day before 15th October is always 14th October).

It is a little ambiguous if the year zero should be a leap year or not, so this is selectable by a #define, however ISO 8601 does define year 0 to be a leap year so that is the default setting.

The allowed range for years is + 2,147,485,547 to - 2,147,481,747, this is limited by the fact that tm\_year in struct tm (which is defined in time.h and so difficult to change without the risk of side effects) may be defined as a 32 bit signed integer. Given this code was written in the year 2022 this is not expected to be an issue for most use cases.

These functions basically ignore leap seconds, but a minute with 61 seconds (00 to 60 rather than 00 to 59) is allowed which is what can happen when a leap second is inserted.

Time zones are supported, but only as an offset from UTC (see %z below). Time zones defined by text (e.g. GMT, see %Z below) can be also used but these functions give no meaning to the actual characters. Summer time is supported by changing the offset from UTC using %z or by using the tm\_isdst field in the struct tm. If this is not done a gap of 1 hour or 1 repeated hour will happen on transitions to/from summer time.

These functions assume the standard C locale, so the days of the week are always named Monday, Tuesday, etc., the months are always named January, February, etc., and AM/PM is used for 12-hour times. Where use in non-English-speaking countries is required numerical forms of dates and times are recommended (see ISO 8601).

These function all use standard C strings; they do not support multibyte character sequences.

# Appendix A –date/time formats for strptime()

This appendix contains the complete list of all possible conversion specifiers for strptime().

Conversion specifiers follow a % sign, those defined are:

%a The day of the week (Monday, Tuesday,...); either the abbreviated (3 character) or full name may be specified. Both upper and lower case is allowed.

%A Equivalent to %a.

%b The month (January, ...) ; either the abbreviated (3 characters) or full name may be specified. Both upper and lower case is allowed.

%B Equivalent to %b

%c date and time [ this is equivalent to %a %b %e %T %Y ]

%C All but the last two digits of the year

Normally used before %y, but can also be used before %g. This means a %C also sets the ISO 8601 century

%d The day of the month [01,31]; leading zeros are permitted but are not required.

%D The date as %m/%d/%y.

%e The day of month (1-31)

%f fraction of a second (the values after the decimal point). The decimal point is implied

(so, needs to be in the format string if it’s actually present)

E.g., "%H:%M:%S.%f" will read 12:59:59.12345

%F Equivalent to %Y-%m-%d (the iso 8601 date format)

%G The ISO 8601 week-based year with century as a decimal number.

The 4-digit year corresponds to the ISO week number (see also %V etc).

This has the same format and value as %Y, except that if the ISO week number belongs to the previous or next year, that year is used instead.

%G (or %C%g or just %g) should only be used with %V and the day of the week (%a,%A,%u,%w)

See also https://en.wikipedia.org/wiki/ISO\_week\_date and https://webspace.science.uu.nl/~gent0113/calendar/isocalendar.htm

%g Replaced by the same year as in %G, but as a decimal number without century (00-99).

%h Equivalent to %b.

%H The hour (24-hour clock) [00,23]; leading zeros are permitted but are not required.

%I The hour (12-hour clock) [01,12]; leading zeros are permitted but are not required.

%j The day number of the year [001,366]; leading zeros are permitted but are not required.

%m The month number [01,12]; leading zeros are permitted but are not required.

%M The minute [00,59]; leading zeros are permitted but are not required.

%n Any white space. Note that a space has the same effect

%p "am" or "pm"

%r 12-hour clock time using the AM/PM notation; equivalent to %I:%M:%S %p

%R The time as %H:%M.

%s seconds since the epoch as a multidigit signed integer.

As this implicitly includes the date and time it should not be used with any other conversion specifiers except perhaps as “%s.%f”.

Limits in years are + 2,147,485,547 to - 2,147,481,747

Note leap seconds are ignored when calculating a date/time from a given number of seconds (mainly as the dates when future leap seconds will be added cannot be predicted).

%S The seconds [00,60]; leading zeros are permitted but are not required.

%t Any white space.

%T The time as %H:%M:%S

%u Weekday as a number 1->7 where Monday=1 and Sunday=7

%U The week number of the year (Sunday as the first day of the week) as a decimal number

[00,53]; leading zeros are permitted but are not required.

If %Y or %C%y or %y together with %U and the day of the week (%a,%A,%u,%w) are all read by the format then a full unique date is defined.

%V The ISO 8601 week number of the year (Monday as the first day of the week) as a

decimal number (01-53).

If the week containing January 1 has four or more days in the new year, then it is week 1; otherwise, it is the last week of the previous year, and the next week is week 1. See %G above for more information.

%w The weekday as a decimal number [0,6], with 0 representing Sunday.

%W The week number of the year (Monday as the first day of the week) as a decimal

number [00,53]; leading zeros are permitted but are not required.

If %Y or %C%y or %y together with %U and the day of the week (%a,%A,%u,%w) are all read by the format then a full unique date is defined.

%x The date, the same as %D.

%X The time, the same as %T

%y The last two digits of the year.

When the format contains neither a C conversion specifier nor a Y conversion specifier, values in the range [69,99] refer to years 1969 to 1999 inclusive and values in the range [00,68] shall refer to years 2000 to 2068 inclusive;

Leading zeros are permitted but are not required.

%Y The full year. Leading zeros are permitted but are not required.

An optional leading sign (+/-) is also allowed. Limits in years are + 2,147,485,547 to - 2,147,481,747.

%z Time zone offset from UTC; a leading plus sign stands for east of UTC,

a minus sign or west of UTC, hours and minutes follow with two digits each and no delimiter between them (as in ISO8601 & common form for RFC 822 date headers). e.g., “-0500” or "+0000". The sign is always required.

%Z time zone name. e.g., “EDT”, "UTC", "GMT","AKST","ET" etc. 2, 3 or 4 letters is required.

% Replaced by %.

Any other character will match the same character in the input.

# Appendix B –date/time formats for strftime()

This appendix contains the complete list of all possible conversion specifiers for strftime().

Conversion specifiers follow a % sign, those defined are:

%a The day of the week (mon, tue,...); abbreviated (3 character).

%A The day of the week (monday, tuesday,...); full name.

%b The month (jan, ...); abbreviated (3 characters).

%B The month (january, ...); full name.

%c date and time [ this is equivalent to %a %b %e %T %Y ]

%C All but the last two digits of the year

Normally used before %y, but can also be used before %g. This means a %C also sets the ISO 8601 century

%d The day of the month [01,31].

%D The date as %m/%d/%y.

%e The day of month (1-31) in a two character field.

%f fraction of a second (the values after the decimal point). The decimal point is implied

(so, needs to be in the format string if it’s actually present)

E.g., "%H:%M:%S.%f" will print 12:59:59.12345

%F Equivalent to %Y-%m-%d (the iso 8601 date format)

%G The ISO 8601 week-based year with century as a decimal number.

The 4-digit year corresponds to the ISO week number (see also %V etc).

This has the same format and value as %Y, except that if the ISO week number belongs to the previous or next year, that year is used instead.

%G (or %C%g or just %g) should only be used with %V and the day of the week (%a,%A,%u,%w)

See also https://en.wikipedia.org/wiki/ISO\_week\_date and https://webspace.science.uu.nl/~gent0113/calendar/isocalendar.htm

%g Replaced by the same year as in %G, but as a decimal number without century (00-99).

%h Equivalent to %b.

%H The hour (24-hour clock) [00,23].

%I The hour (12-hour clock) [01,12].

%j The day number of the year [001,366].

%m The month number [01,12].

%M The minute [00,59].

%n newline character.

%p "am" or "pm"

%r 12-hour clock time using the AM/PM notation; equivalent to %I:%M:%S %p

%R The time as %H:%M.

%s seconds since the epoch as a multidigit signed integer.

As this implicitly includes the date and time it should not be used with any other conversion specifiers except perhaps as “%s.%f”.

Limits in years are + 2,147,485,547 to - 2,147,481,747

Note leap seconds are ignored when calculating a date/time from a given number of seconds (mainly as the dates when future leap seconds will be added cannot be predicted).

%S The seconds [00,60].

%t Tab character.

%T The time as %H:%M:%S

%u Weekday as a number 1->7 where Monday=1 and Sunday=7

%U The week number of the year (Sunday as the first day of the week) as a decimal number

[00,53].

If %Y or %C%y or %y together with %U and the day of the week (%a,%A,%u,%w) are all printed by the format then a full unique date is defined.

%V The ISO 8601 week number of the year (Monday as the first day of the week) as a

decimal number (01-53).

If the week containing January 1 has four or more days in the new year, then it is week 1; otherwise, it is the last week of the previous year, and the next week is week 1. See %G above for more information.

%w The weekday as a decimal number [0,6], with 0 representing Sunday.

%W The week number of the year (Monday as the first day of the week) as a decimal

number [00,53].

If %Y or %C%y or %y together with %U and the day of the week (%a,%A,%u,%w) are all printed by the format then a full unique date is defined.

%x The date, the same as %D.

%X The time, the same as %T

%y The last two digits of the year.

When the format contains neither a C conversion specifier nor a Y conversion specifier, values in the range [69,99] refer to years 1969 to 1999 inclusive and values in the range [00,68] shall refer to years 2000 to 2068 inclusive.

%Y The full year.

A leading -ve sign (-) is output if required. Limits in years are + 2,147,485,547 to - 2,147,481,747.

%z Time zone offset from UTC; a leading plus sign stands for east of UTC,

a minus sign or west of UTC, hours and minutes follow with two digits each and no delimiter between them (as in ISO8601 & common form for RFC 822 date headers). e.g., “-0500” or "+0000". The sign is always printed.

%Z time zone name. e.g., “EDT”, "UTC", "GMT","AKST","ET" etc. 2, 3 or 4 letters as

required.

% Replaced by %.

Any other character is transferred directly to the output string.

# History

Strftime.c was created by starting from strftime.c obtained from [https://github.com/arnoldrobbins/strftime](https://github.com/arnoldrobbins/strftime%20on%203/4/2022) .

Thanks to Arnold Robbins and the other contributors to the original file (ado@elsie.nci.nih.gov,

Tor Lillqvist [tml@tik.vtt.fi](mailto:tml@tik.vtt.fi), [chip@chinacat.unicom.com](mailto:chip@chinacat.unicom.com) and Tanaka Akira [akr@m17n.org](mailto:akr@m17n.org)).

Strptime.c This file was created by starting from <https://stackoverflow.com/questions/667250/strptime-in-windows> .

Thanks to K. Shepherd who supplied the original stackoverflow answer and jjv360, Arno Duvenhage, ryyker & Gebi Miguel who commented on it.

The implementations here are now significantly different from the original versions, and this release comes with a comprehensive test program (currently with 7300275 tests) which should give high confidence in the code.

# Changes

1v0 - 3/1/2021 - 1st release on Github

1v1 – 25/5/2022 – (this) manual added.

- main functions now start with ya\_ to avoid conflicts with versions in standard libraries (especially when used from C++).