Rocky Enterprise Linux 9.2 Manual Pages on command 'tzfile.5'

$ man tzfile.5

TZFILE(5)   Linux Programmer's Manual   TZFILE(5)

NAME

   tzfile - timezone information

DESCRIPTION

   The timezone information files used by tzset(3) are typically found un?

   der a directory with a name like /usr/share/zoneinfo. These files use

   the format described in Internet RFC 8536. Each file is a sequence of

   8-bit bytes. In a file, a binary integer is represented by a sequence

   of one or more bytes in network order (bigendian, or high-order byte

   first), with all bits significant, a signed binary integer is repre?

   sented using two's complement, and a boolean is represented by a one-

   byte binary integer that is either 0 (false) or 1 (true).   The format

   begins with a 44-byte header containing the following fields:

   * The magic four-byte ASCII sequence ?TZif? identifies the file as a

     timezone information file.

   * A byte identifying the version of the file's format (as of 2017, ei?

     ther an ASCII NUL, or ?2?, or ?3?).

   * Fifteen bytes containing zeros reserved for future use.

* Six four-byte integer values, in the following order:

 tzh_ttisutcnt

> The number of UT/local indicators stored in the file. (UT is
>
> Universal Time.)

 tzh_ttisstdcnt

> The number of standard/wall indicators stored in the file.

 tzh_leapcnt

> The number of leap seconds for which data entries are stored
>
> in the file.

 tzh_timecnt

> The number of transition times for which data entries are
>
> stored in the file.

 tzh_typecnt

> The number of local time types for which data entries are
>
> stored in the file (must not be zero).

 tzh_charcnt

> The number of bytes of time zone abbreviation strings stored
>
> in the file.

The above header is followed by the following fields, whose lengths de?
pend on the contents of the header:

* tzh_timecnt four-byte signed integer values sorted in ascending or?
 der. These values are written in network byte order. Each is used
 as a transition time (as returned by time(2)) at which the rules for
 computing local time change.

* tzh_timecnt one-byte unsigned integer values; each one but the last
 tells which of the different types of local time types described in
 the file is associated with the time period starting with the same-
 indexed transition time and continuing up to but not including the
 next transition time. (The last time type is present only for con?
 sistency checking with the POSIX-style TZ string described below.)
 These values serve as indices into the next field.

* tzh_typecnt ttinfo entries, each defined as follows:

> struct ttinfo {

```
        int32_t      tt_utoff;

        unsigned char  tt_isdst;

        unsigned char  tt_desigidx;

   };
```

Each structure is written as a four-byte signed integer value for

tt_utoff, in network byte order, followed by a one-byte boolean for

tt_isdst and a one-byte value for tt_desigidx.   In each structure,

tt_utoff gives the number of seconds to be added to UT, tt_isdst

tells whether tm_isdst should be set by localtime(3) and tt_desigidx

serves as an index into the array of time zone abbreviation bytes

that follow the ttinfo structure(s) in the file. The tt_utoff value

is never equal to -2**31, to let 32-bit clients negate it without

overflow. Also, in realistic applications tt_utoff is in the range

[-89999, 93599] (i.e., more than -25 hours and less than 26 hours);

this allows easy support by implementations that already support the

POSIX-required range [-24:59:59, 25:59:59].

* tzh_leapcnt pairs of four-byte values, written in network byte order;

  the first value of each pair gives the nonnegative time (as returned

  by time(2)) at which a leap second occurs; the second is a signed in?

  teger specifying the total number of leap seconds to be applied dur?

  ing the time period starting at the given time. The pairs of values

  are sorted in ascending order by time. Each transition is for one

  leap second, either positive or negative; transitions always sepa?

  rated by at least 28 days minus 1 second.

* tzh_ttisstdcnt standard/wall indicators, each stored as a one-byte

  boolean; they tell whether the transition times associated with local

  time types were specified as standard time or local (wall clock)

  time.

* tzh_ttisutcnt UT/local indicators, each stored as a one-byte boolean;

  they tell whether the transition times associated with local time

  types were specified as UT or local time. If a UT/local indicator is

  set, the corresponding standard/wall indicator must also be set.

The standard/wall and UT/local indicators were designed for transform?

ing a TZif file's transition times into transitions appropriate for an‐

other time zone specified via a POSIX-style TZ string that lacks rules.

For example, when TZ="EET-2EEST" and there is no TZif file "EET-2EEST",

the idea was to adapt the transition times from a TZif file with the

well-known name "posixrules" that is present only for this purpose and

is a copy of the file "Europe/Brussels", a file with a different UT

offset. POSIX does not specify this obsolete transformational behav‐

ior, the default rules are installation-dependent, and no implementa‐

tion is known to support this feature for timestamps past 2037, so

users desiring (say) Greek time should instead specify TZ="Eu‐

rope/Athens" for better historical coverage, falling back on

TZ="EET-2EEST,M3.5.0/3,M10.5.0/4" if POSIX conformance is required and

older timestamps need not be handled accurately.

The localtime(3) function normally uses the first ttinfo structure in

the file if either tzh_timecnt is zero or the time argument is less

than the first transition time recorded in the file.

Version 2 format

For version-2-format timezone files, the above header and data are fol‐

lowed by a second header and data, identical in format except that

eight bytes are used for each transition time or leap second time.

(Leap second counts remain four bytes.) After the second header and

data comes a newline-enclosed, POSIX-TZ-environment-variable-style

string for use in handling instants after the last transition time

stored in the file or for all instants if the file has no transitions.

The POSIX-style TZ string is empty (i.e., nothing between the newlines)

if there is no POSIX representation for such instants. If nonempty,

the POSIX-style TZ string must agree with the local time type after the

last transition time if present in the eight-byte data; for example,

given the string ?WET0WEST,M3.5.0,M10.5.0/3? then if a last transition

time is in July, the transition's local time type must specify a day‐

light-saving time abbreviated ?WEST? that is one hour east of UT.

Also, if there is at least one transition, time type 0 is associated

with the time period from the indefinite past up to but not including

the earliest transition time.

Version 3 format

For version-3-format timezone files, the POSIX-TZ-style string may use two minor extensions to the POSIX TZ format, as described in newtzset(3). First, the hours part of its transition times may be signed and range from -167 through 167 instead of the POSIX-required unsigned values from 0 through 24. Second, DST is in effect all year if it starts January 1 at 00:00 and ends December 31 at 24:00 plus the difference between daylight saving and standard time.

Interoperability considerations

Future changes to the format may append more data.

Version 1 files are considered a legacy format and should be avoided, as they do not support transition times after the year 2038. Readers that only understand Version 1 must ignore any data that extends beyond the calculated end of the version 1 data block.

Writers should generate a version 3 file if TZ string extensions are necessary to accurately model transition times. Otherwise, version 2 files should be generated.

The sequence of time changes defined by the version 1 header and data block should be a contiguous subsequence of the time changes defined by the version 2+ header and data block, and by the footer. This guide? line helps obsolescent version 1 readers agree with current readers about timestamps within the contiguous subsequence. It also lets writ? ers not supporting obsolescent readers use a tzh_timecnt of zero in the version 1 data block to save space.

Time zone designations should consist of at least three (3) and no more than six (6) ASCII characters from the set of alphanumerics, ?-?, and ?+?. This is for compatibility with POSIX requirements for time zone abbreviations.

When reading a version 2 or 3 file, readers should ignore the version 1 header and data block except for the purpose of skipping over them. Readers should calculate the total lengths of the headers and data blocks and check that they all fit within the actual file size, as part

of a validity check for the file.

Common interoperability issues

This section documents common problems in reading or writing TZif files. Most of these are problems in generating TZif files for use by older readers. The goals of this section are:

* to help TZif writers output files that avoid common pitfalls in older or buggy TZif readers,

* to help TZif readers avoid common pitfalls when reading files gener? ated by future TZif writers, and

* to help any future specification authors see what sort of problems arise when the TZif format is changed.

When new versions of the TZif format have been defined, a design goal has been that a reader can successfully use a TZif file even if the file is of a later TZif version than what the reader was designed for. When complete compatibility was not achieved, an attempt was made to limit glitches to rarely used timestamps, and to allow simple partial workarounds in writers designed to generate new-version data useful even for older-version readers. This section attempts to document these compatibility issues and workarounds, as well as to document other common bugs in readers.

Interoperability problems with TZif include the following:

* Some readers examine only version 1 data. As a partial workaround, a writer can output as much version 1 data as possible. However, a reader should ignore version 1 data, and should use version 2+ data even if the reader's native timestamps have only 32 bits.

* Some readers designed for version 2 might mishandle timestamps after a version 3 file's last transition, because they cannot parse exten? sions to POSIX in the TZ-like string. As a partial workaround, a writer can output more transitions than necessary, so that only far-future timestamps are mishandled by version 2 readers.

* Some readers designed for version 2 do not support permanent daylight saving time, e.g., a TZ string ?EST5EDT,0/0,J365/25? denoting perma? nent Eastern Daylight Time (-04). As a partial workaround, a writer

can substitute standard time for the next time zone east, e.g.,

?AST4? for permanent Atlantic Standard Time (-04).

* Some readers ignore the footer, and instead predict future timestamps

  from the time type of the last transition. As a partial workaround,

  a writer can output more transitions than necessary.

* Some readers do not use time type 0 for timestamps before the first

  transition, in that they infer a time type using a heuristic that

  does not always select time type 0. As a partial workaround, a

  writer can output a dummy (no-op) first transition at an early time.

* Some readers mishandle timestamps before the first transition that

  has a timestamp not less than $-2^{**}31$. Readers that support only

  32-bit timestamps are likely to be more prone to this problem, for

  example, when they process 64-bit transitions only some of which are

  representable in 32 bits. As a partial workaround, a writer can out?

  put a dummy transition at timestamp $-2^{**}31$.

* Some readers mishandle a transition if its timestamp has the minimum

  possible signed 64-bit value. Timestamps less than $-2^{**}59$ are not

  recommended.

* Some readers mishandle POSIX-style TZ strings that contain ?<? or

  ?>?. As a partial workaround, a writer can avoid using ?<? or ?>?

  for time zone abbreviations containing only alphabetic characters.

* Many readers mishandle time zone abbreviations that contain non-ASCII

  characters. These characters are not recommended.

* Some readers may mishandle time zone abbreviations that contain fewer

  than 3 or more than 6 characters, or that contain ASCII characters

  other than alphanumerics, ?-?, and ?+?. These abbreviations are not

  recommended.

* Some readers mishandle TZif files that specify daylight-saving time

  UT offsets that are less than the UT offsets for the corresponding

  standard time. These readers do not support locations like Ireland,

  which uses the equivalent of the POSIX TZ string

  ?IST-1GMT0,M10.5.0,M3.5.0/1?, observing standard time (IST, +01) in

  summer and daylight saving time (GMT, +00) in winter. As a partial

workaround, a writer can output data for the equivalent of the POSIX TZ string ?GMT0IST,M3.5.0/1,M10.5.0?, thus swapping standard and day?light saving time. Although this workaround misidentifies which part of the year uses daylight saving time, it records UT offsets and time zone abbreviations correctly.

Some interoperability problems are reader bugs that are listed here mostly as warnings to developers of readers.

* Some readers do not support negative timestamps. Developers of dis?tributed applications should keep this in mind if they need to deal with pre-1970 data.

* Some readers mishandle timestamps before the first transition that has a nonnegative timestamp. Readers that do not support negative timestamps are likely to be more prone to this problem.

* Some readers mishandle time zone abbreviations like ?-08? that con?tain ?+?, ?-?, or digits.

* Some readers mishandle UT offsets that are out of the traditional range of -12 through +12 hours, and so do not support locations like Kiritimati that are outside this range.

* Some readers mishandle UT offsets in the range [-3599, -1] seconds from UT, because they integer-divide the offset by 3600 to get 0 and then display the hour part as ?+00?.

* Some readers mishandle UT offsets that are not a multiple of one hour, or of 15 minutes, or of 1 minute.

SEE ALSO

time(2), localtime(3), tzset(3), tzselect(8), zdump(8), zic(8).

Olson A, Eggert P, Murchison K. The Time Zone Information Format (TZif). 2019 Feb. Internet RFC 8536 ?https://www.rfc-editor.org/info/rfc8536? doi:10.17487/RFC8536 ?https://doi.org/10.17487/RFC8536?.

COLOPHON

This page is part of release 5.10 of the Linux man-pages project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at https://www.kernel.org/doc/man-pages/.