NAME

tzfile - timezone information

DESCRIPTION

The timezone information files used by tzset(3) are found under */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 represented 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 2021, either an ASCII NUL, "2", "3", or "4").
- 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 depend on the contents of the header:

tzh_timecnt

four-byte signed integer values sorted in ascending order. These values are written in network byte order. Each is used as a transition time (as returned by 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 consistency 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* entries in the file; if the designated string is "00", the *ttinfo* entry is a placeholder indicating that local time is unspecified. 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_charcnt

bytes that represent time zone designations, which are null-terminated byte strings, each indexed by the *tt_desigidx* values mentioned above. The byte strings can overlap if one is a suffix of the other. The encoding of these strings is not specified.

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(3)) at which a leap second occurs or at which the leap second table expires; the second is a signed integer specifying the correction, which is the *total* number of leap seconds to be applied during the time period starting at the given time. The pairs of values are sorted in strictly ascending order by time. Each pair denotes one leap second, either positive or negative, except that if the last pair has the same correction as the previous one, the last pair denotes the leap second table's expiration time. Each leap second is at the end of a UTC calendar month. The first leap second has a nonnegative occurrence time, and is a positive leap second if and only if its correction is positive; the correction for each leap second after the first differs from the previous leap second by either 1 for a positive leap second, or -1 for a negative leap second. If the leap second table is empty, the leap-second correction is zero for all timestamps; otherwise, for timestamps before the first occurrence time, the leap-second correction is zero if the first pair's correction is 1 or -1, and is unspecified otherwise (which can happen only in files truncated at the start).

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 transforming a TZif file's transition times into transitions appropriate for another time zone specified via a POSIX-style TZ string that lacks rules. For example, when TZ="EET2EEST" and there is no TZif file "EET2EEST", 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 behavior, the default rules are installation-dependent, and no implementation is known to support this feature for timestamps past 2037, so users desiring (say) Greek time should instead specify TZ="Europe/Athens" for better historical coverage, falling back on TZ="EET2EEST,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 followed 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-style 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,M3.5.0/1,M10.5.0" then if a last transition time is in July, the transition’s local time type must specify a daylight-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.

Version 4 format

For version-4-format TZif files, the first leap second record can have a correction that is neither +1 nor -1, to represent truncation of the TZif file at the start. Also, if two or more leap second transitions are present and the last entry’s correction equals the previous one, the last entry denotes the expiration of the leap second table instead of a leap second; timestamps after this expiration are unreliable in that future releases will likely add leap second entries after the expiration, and the added leap seconds will change how post-expiration timestamps are treated.

Interoperability considerations

Future changes to the format may append more data.

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

Other than version 1, writers should generate the lowest version number needed by a file’s data. For example, a writer should generate a version 4 file only if its leap second table either expires or is truncated at the start. Likewise, a writer not generating a version 4 file should generate a version 3 file only if TZ string extensions are necessary to accurately model transition times.

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

When a TZif file contains a leap second table expiration time, TZif readers should either refuse to process post-expiration timestamps, or process them as if the expiration time did not exist (possibly with an error indication).

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 higher 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.

When a positive leap second occurs, readers should append an extra second to the local minute containing the second just before the leap second. If this occurs when the UTC offset is not a multiple of 60 seconds, the leap second occurs earlier than the last second of the local minute and the minute's remaining local seconds are numbered through 60 instead of the usual 59; the UTC offset is unaffected.

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 generated 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 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 or higher file's last transition, because they cannot parse extensions 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 with transitions after 24:00 - e.g., a TZ string "EST5EDT,0/0,J365/25" denoting permanent Eastern Daylight Time (-04). As a workaround, a writer can substitute standard time for two time zones east, e.g., "XXX3EDT4,0/0,J365/23" for a time zone with a never-used standard time (XXX, -03) and negative daylight saving time (EDT, -04) all year. Alternatively, 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 designed for version 2 or 3, and that require strict conformance to RFC 8536, reject version 4 files whose leap second tables are truncated at the start or that end in expiration times.

- 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 output 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 "IST1GMT0,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 daylight 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 readers generate ambiguous timestamps for positive leap seconds that occur when the UTC offset is not a multiple of 60 seconds. For example, in a timezone with UTC offset +01:23:45 and with a positive leap second 78796801 (1972-06-30 23:59:60 UTC), some readers will map both 78796800 and 78796801 to 01:23:45 local time the next day instead of mapping the latter to 01:23:46, and they will map 78796815 to 01:23:59 instead of to 01:23:60. This has not yet been a practical problem, since no civil authority has observed such UTC offsets since leap seconds were introduced in 1972.

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 distributed 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 contain "+", "", 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(3), localtime(3), tzset(3), tzsetup(8), zic(8), zdump(8)

A. Olson, P. Eggert, and K. Murchison, *The Time Zone Information Format (TZif)*, RFC 8536, https://datatracker.ietf.org/doc/html/rfc8536, https://doi.org/10.17487/RFC8536, February 2019.