TZDB and Some Challenges of Long Data

Paul R. Eggert
UCLA Computer Science Dept.
The Time Zone Database (TZDB)

- Contains the history of the world’s civil time since 1970
- Mostly public domain since its inception
- A bit of code was derived from BSD and is Berkeley-licensed
- Contributors are unpaid volunteers
- Development is (mostly) public
- You most likely have a copy of it in your cell phone, laptop, Docker container, etc.
The civil time problem

- Governments set civil time as offset from Universal Time.
- Governments sometimes have daylight-saving time rules that change UT offsets, typically by one hour twice a year.
- Governments sometimes change their minds, and alter UT offsets or DST rules.
- Civil timestamps need to be converted from one locality to another, for any timestamp in the past or future.
What Unix got right about time

- Credit Dennis Ritchie and Ken Thompson, who won the Turing Award for their development of Unix.
- Developed by Bell Labs (AT&T) originally for internal use, when antitrust laws prevented AT&T from entering the computer business.
- AT&T’s network crossed time zone boundaries.
- Ritchie and Thompson’s system therefore did not run on local time; it ran on Universal Time.
Unix's time model

- Unix (7th edition, 1979) was the first readily portable version, with a kernel model still widely used.
- Its ftime system call yielded four numbers:
  - Number of seconds since 1970-01-01 00:00:00 UTC
  - Number of milliseconds since the start of the second
  - UT offset of this system, in seconds
  - Whether daylight saving time is ever observed on this system
Unix v7 civil time problems

- To change the time zone, or whether DST is ever observed, you needed to recompile the kernel and reboot.
- Time zone and DST flag were system-wide.
- Daylight-saving rules were hard-coded in the C library.
- If Congress changed the rules or your country used different rules, you needed to recompile the C library and relink all your applications.
Unix System V fixed an easy part

- Kernel keeps track only of UT.
- Each process has its own time zone, specified by the TZ environment variable.
- TZ='PST8PDT' means, “The UT offset is 8 hours west of Greenwich, DST is observed, and the time zone abbreviation is ‘PST’ or ‘PDT’ depending on whether DST is in effect.”
- The C library’s localtime function obeys the TZ value.
- BUT: DST rules still hard-coded in the C library.
- SO: chaos outside the US.
The Open Group weighs in

- The standardization committee for POSIX, the Portable Operating System Interface (first edition, 1988).
- Generalized TZ values to specify daylight saving rules.
- TZ='PST8:00PDT7:00,M3.2.0/2:00,M11.1.0/2:00' for California.
- Usually abbreviated to TZ='PST8PDT,M3.2.0,M11.1.0', as POSIX defaults to US-like DST transition times and offsets.
- A pain to get right; plus, what happens to old timestamps when Congress changes the DST rules?
• In 1986, National Institutes of Health developer Arthur David Olson wrote a modified C library and time zone compiler zic.
• Central innovation: move the DST rules out of the C library, and into data files.
• Timestamp conversion must be fast, though.
• So the runtime data files are in a new binary format TZif.
• zic compiles source data (.zi files) into TZif format.
• Adopted by Sun Microsystems in 1986, spread elsewhere.
• Limited coverage of world locations and timestamps.
In 1989 I co-founded a software and network consulting company Twin Sun, Inc.

Most customers were based in Japan, with clients scattered around east Asia, all needing accurate time.

The NIH database had one rule for Japan, “Zone Japan 9:00 – JST” and similarly for South Korea, Taiwan but little else.

I started filling in the blanks for east Asia, and decided to finish the job for the world.

“How hard could it be?”, thought I.
An assist from the UCLA Library

- As a Life Member of the UCLA Alumni Association, I had a lifetime UCLA library card.
- I went to the library and found that the main people who cared about time zone information were astrologers!
- I found several astrology books, the most detailed of which were by Thomas Shanks.
Thomas Shanks and time zones

- Born in Lima, Ohio on April 9, 1942 at 2:55 EWT.
- He gained computer skills, researched “the factual basis of astrological assertions”, and never got his humanities PhD.
- Birth certificates use local time, but astrologers need UT.
- Shanks wrote software to convert timestamps.
- He gathered announcements from newspapers in Ohio, Indiana, etc. to find out when each time changed its clocks, a common occurrence in the US.
- Back in the 1970s, though, most astrologers didn’t have computers.
What I got from Shanks’s books

• “How hard can it be?” - too much work for a volunteer.
• I got the gist of a small subset of Shanks’s books, and manually created TZDB-style tables for the rest of the world.
• The TZDB naming convention (e.g., “Japan”) was wrong, as Shanks said Japan had multiple time zones in the past.
• I needed names for subdivisions of countries, but province/district names are obscure and boundaries change.
• Continent/City names (e.g., “Asia/Tokyo”) avoid politics.
There are now 100 lines of source code just for Japan now. Most of them are commentary, giving citations like “the Meiji Emperor announced Ordinance No. 167 of Meiji Year 28 ‘The clause about standard time’ ... The adoption began from Jan 1, 1896. [http://ja.wikisource.org/wiki/ 標準時ニ関スル件 _ (公布時 )”]

Shanks is no longer cited, because he was mostly wrong about Japan.
A simple TZDB entry today (2)

# Rule NAME FROM TO - IN ON AT SAVE
Rule Japan 1948 only - May Sat>=1 24:00 1:00 D
Rule Japan 1948 1951 - Sep Sat>=8 25:00 0 S
Rule Japan 1949 only - Apr Sat>=1 24:00 1:00 D
Rule Japan 1950 1951 - May Sat>=1 24:00 1:00 D

# Zone NAME STDOFF RULES FORMAT UNTIL
Zone Asia/Tokyo 9:18:59 – LMT 1887 Dec 31 15:00u 9:00 Japan J%sT
The problem of prediction

- Morocco is at UT+01 normally, UT+00 during Ramadan.
- The Islamic calendar depends on actual observations of the moon; these can be predicted but not entirely accurately.
- The Moroccan government announces DST rules a few weeks before Ramadan, with slop before and after.
- But TZDB’s .zi format relies on Gregorian calendar.
The book “Calendrical Calculations” by Ed Reingold and Nachum Dershowitz is the definitive source on calendrical conversions.

I used its calculation code (written in Lisp and ported to GNU Emacs Lisp) to predict Ramadan+governmental slop for Morocco.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Morocco</th>
<th>Year</th>
<th>Type</th>
<th>Start Month</th>
<th>Start Day</th>
<th>Start Time</th>
<th>End Month</th>
<th>End Day</th>
<th>End Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule</td>
<td>Morocco</td>
<td>2023</td>
<td>only</td>
<td>Mar</td>
<td>19</td>
<td>3:00</td>
<td>-1:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rule</td>
<td>Morocco</td>
<td>2023</td>
<td>only</td>
<td>Apr</td>
<td>30</td>
<td>2:00</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rule</td>
<td>Morocco</td>
<td>2024</td>
<td>only</td>
<td>Mar</td>
<td>10</td>
<td>3:00</td>
<td>-1:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rule</td>
<td>Morocco</td>
<td>2024</td>
<td>only</td>
<td>Apr</td>
<td>14</td>
<td>2:00</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Lisp is not enough for prediction

- It’s not just that we can’t predict the Moon’s location many years in the future; we can’t predict lunar observations, which are partly political.
- Governments sometimes change the rules with little notice (even less than a day).
- Governments sometimes change the rules and then change back before the changes take effect.
- We need to update TZDB instances in billions of devices.
Updating billions of devices

• We issue a new TZDB release, trust downstream distributors to do the right thing.
• Some take their time to repackage it, but give users the ability to download from us directly if needed.
• Android originally copied the Linux model, but this meant changes required upgrade to new Android version and reboot. They evolved to something less drastic.
• Many people don’t update their cell phones, though.
The astrologers strike back

- After Shanks passed away, his estate eventually sold his proprietary data.
- New owners were not happy about TZDB.
- NIH pulled TZDB off the Internet.
EFF to the rescue

- Electronic Frontier Foundation pledged legal support.
- They found first-class attorneys to defend us in Boston.
- This was not sheer luck: I knew about *Feist* in the early 1990s when I started contributing to TZDB.
- I read Shanks’s books and interpreted his data; I never had or ran Shanks’s programs, which are still proprietary.
IANA to the rescue too

- TZDB briefly moved to Australia during the ruckus.
- Soon, IANA volunteered to host it at https://iana.org/tz.
- I co-wrote RFCs and am now the IESG-designated TZ Primary Coordinator, since Olson has retired.
- According to the rules, the IESG can designate a different coordinator if I screw up or get hit by a bus (or both).
- We have a backup coordinator Tim Parenti (CMU) in case that happens.
I made two mistakes when taking data from Shanks. The first was to trust his data, which he often invented. The second was to adapt his lookup method, which was based on country+city. I wrote a simple auxiliary table zone.tab to help people find Zone names. Sample entry: “JP +353916+1394441 Asia/Tokyo”. “How hard could this be?”, I thought.
Political problems (2)

- In hindsight adding politics was my biggest mistake.
- In some cases (e.g., Kashmir) the bullet was dodged by needing no Zones.
- In others (e.g., Europe/Simferopol) we were not so lucky.
- And in still others (e.g, Europe/Kiev) there are disputes about English spellings of city names.
Fixing political problems is its own political problem.
I am coalescing Zones that I mistakenly created in the 1990s because of Shanks, that were put in only for political reasons.
Although I preserved old data in a ‘backzone’ file, the change has been controversial in the TZDB community, and I almost lost my TZDB job last year due to the controversy.
TZDB’s future

- TZDB fixed most signed 32-bit (Y2038) problems long ago.
- Distribution via GNU Tar has Y2242 problems, fixed via `--format=pax -pax-option='delete=atime,delete=ctime'`.
- For 32-bit int, 64-bit time_t platforms, the standard C API stops working after the year 285,428,681 ($2^{53} - 1 + 1900$).
- The length of day (LOD) will exceed 25 hours by that year.
- In the shorter term, political problems could well kill off TZDB one way or another.