Time Standards: UTC, GPS, TAI, and Unix-C times.

0

The relationship between different time-standards and labels is somewhat complex. The following web sites contains further information about these different time standards:
`ftp://tycho.usno.navy.mil/pub/series/ser14.txt` ftp://tycho.usno.navy.mil/pub/series/ser14.txt
`ftp://maia.usno.navy.mil/ser7/tai-utc.dat` ftp://maia.usno.navy.mil/ser7/tai-utc.dat
`http://tycho.usno.navy.mil/time.html` . http://tycho.usno.navy.mil/time.html
What is presented here is merely a summary for the perplexed. The situation is complicated slightly because on standard Unix machines (Solaris 2.6, Linux 2.0*, etc) the time functions in the ``ANSI-standard C library" do not behave according to the rather vaguely-defined ANSI/POSIX standards. (In particular, the `gmtime()` function in the C standard library does not know about leap-seconds - although the documentation generally suggests that it should!)

Let us begin by defining a physical time t, which is the time coordinate used in physics lectures. It advances linearly, completing each unit step at the same instant that a perfect pendulum with a frequency of 1 Hz completes a new oscillation. Without loss of generality we will set $t=0$ at the stroke of midnight, January 1, 1970 UTC. This time is also denoted Jan 1 00:00:00 UTC. Note that we will not discuss *local* time in this section, which differs from UTC by a fixed number of hours that depend upon your time-zone and upon whether or not daylight savings time is in effect. We will assume that once you know the UTC time, you can do the necessary addition or subtraction yourself, to determine local time at any point of interest on the earth.

Universal Coordinated Time (UTC), Global Positioning System (GPS) and International Atomic Time (TAI) are all well-defined global standards. By Unix-C time we mean here the value of t. This is what is returned by the Standard C-library `time()` function, which advances its output by one every second, starting from Jan 1 00:00:00 UTC, *provided that the computer is started at Jan 1 00:00:00 UTC and runs continuously and without error after that*. (See below for an explanation of why this qualification is required!)

UTC should be thought of as a system for attaching "human readable" labels such as ``March 11, 1983, at 12:10" to particular moments in time. (We use ``military" or 24-hour time to distinguish am and pm.) This is done by advancing in the standard pattern of 1 hour every 3600 seconds, one day every 24 hours, and so on, with two exceptions. These exceptions are necessary because if they were not made, eventually people in the northern hemisphere would be suffering snowfalls in August.

The first of the two exceptions is easy. Once every four years, according to the pattern 1972, 1976, ... there is a leap year, which is a year containing one extra day. In these years, February has 29 days not 28. This affects the pattern by which UTC time follows t, and is a minor nuisance. However, because it follows a regular deterministic pattern, leap years with their extra day present no real problems.

The real complications arise because the earth is gradually slowing in its rotation about its axis, from the effects of earth-moon and earth-sun tidal friction, and because the sunspot cycles affects the heating of the earth and thus its mass distribution and moment of inertia. These effects are not easily predicted in advance, and thus on a regular basis (between once every two years and twice a year) an extra second is inserted into the UTC label. The decision about when to do this is made by the International Earth Rotation Service (IERS) `http://hpiers.obspm.fr/`. http://hpiers.obspm.fr/ This one extra second is generally added right after what would be the final second of a month (generally December or June). Thus, at these times, the normal pattern of ..., 23:59:58, 23:59:59, 00:00:00 ...is broken and replaced by ..., 23:59:58, 23:59:59, 23:59:60, 00:00:00, ... The next leap second will be inserted into UTC in this way at the end of December 1998. In principle, a leap second can be either positive or negative, although there have not yet (as of September 1998) been any negative leap seconds.

Unfortunately, there is one additional small complication. Until Jan 1, 1972 UTC, the duration of one UTC second was not equal to the duration of one (physical, TAI=GPS=CTime) second. This can be seen most easily from `ftp://maia.usno.navy.mil/ser7/tai-utc.dat` ftp://maia.usno.navy.mil/ser7/tai-utc.dat. Here is a small extract from that file.

```
...
 1966 JAN  1 =JD 2439126.5  TAI-UTC=   4.3131700 S + (MJD - 39126.) X 0.002592 S
 1968 FEB  1 =JD 2439887.5  TAI-UTC=   4.2131700 S + (MJD - 39126.) X 0.002592 S
 1972 JAN  1 =JD 2441317.5  TAI-UTC=  10.0       S + (MJD - 41317.) X 0.0      S
 1972 JUL  1 =JD 2441499.5  TAI-UTC=  11.0       S + (MJD - 41317.) X 0.0      S
 1973 JAN  1 =JD 2441683.5  TAI-UTC=  12.0       S + (MJD - 41317.) X 0.0      S
...
```

Here, the Modified Julian Day (MJD) is defined as follows: MJD = JD - 2400000.5, where the Julian Day increments by one at noon every day. Notice that these relationships imply that, before January 1, 1972 UTC, the difference between atomic time (TAI) and UTC time is *not* an integer number of seconds!

The values of JD and MJD at some interesting times are given below

```
1968 Feb 1 00:00:00 UTC JD=2439887.5 MJD=39887
1969 Feb 1 00:00:00 UTC JD=2440253.5 MJD=40253
1970 Jan 1 00:00:00 UTC JD=2440587.5 MJD=40587
1970 Feb 1 00:00:00 UTC JD=2440618.5 MJD=40618
1971 Jan 1 00:00:00 UTC JD=2440952.5 MJD=40952
1972 Jan 1 00:00:00 UTC JD=2441317.5 MJD=41317
```

In particular, on Jan 1 00:00:00 1970 UTC, the formula above gives:

$$t = 63072002 \tag{17.0.335}$$

We will ignore the 82 microseconds in what follows.

Much of the complication arises because the standard Unix C-library function `gmtime()`, which takes as its argument the number of seconds after Jan 1, 1970 UTC, and should return the UTC time, *does not return the correct UTC time*. *The function `gmtime()` fails to take account of leap seconds*. The relationship between t, Unix-C time, and UTC time is demonstrated in the table below, which shows the effects of the leap seconds and the erroneous behavior of `gmtime()`. Note that in this table, the definition of ``leap second" is not made precise until on or after Jan 1, 1972 UTC. Until that time, the number of leap seconds can be non-integer: here we have assumed that it is integer. (This is of no consequence provided that we only study the relationship between our different time standards for times after Jan 1, 1972 UTC).

Table: Relationship between different time coordinates. The UTC time should be thought of as a ``human readable" label that gets attached to the time quantities. The number of leap seconds is often denoted by TAI-UTC: it is equal to the amount by which the Unix notion of UTC differs from the actual time: TAI-UTC=`gmtime(time())`-UTC+8 sec. The most severe problem with the Unix notion of time is that in actually setting the time on an individual machine, one sets the `time()` function to return the wrong value. Thus, at physical time $t = \mathrm{GPS} + 315964811\ \mathrm{sec}$ if you correctly set the machine time to Jan 1 00:00:00 1972 UTC, and then immediately call the `time()` function, it will (incorrectly) return the value 63072000. (Note that this table incorrectly treats TAI-UTC as an integer before Jan 1, 1972 UTC.)

t physical	GPS	Unix-C	UTC time label	`gmtime()`	Leap sec
(sec)	time (sec)	`time()`		function of `time()`	TAI-UTC
0	-315964811	0	Jan 1 00:00:00 1970 UTC	Jan 1 00:00:00 1970	8
1	-315964810	1	Jan 1 00:00:01 1970 UTC	Jan 1 00:00:01 1970	8
...					
33350399	-282614412	33350399	Jan 21 23:59:59 1971 UTC	Jan 21 23:59:59 1971	8
33350400	-282614411	33350400	Jan 21 23:59:60 1971 UTC	Jan 22 00:00:00 1971	8
33350401	-282614410	33350401	Jan 22 00:00:00 1972 UTC	Jan 22 00:00:01 1971	9
...					
63071999	-252892812	63071999	Dec 31 23:59:58 1971 UTC	Dec 31 23:59:59 1971	9
63072000	-252892811	63072000	Dec 31 23:59:59 1971 UTC	Jan 1 00:00:00 1972	9
63072001	-252892810	63072001	Dec 31 23:59:60 1971 UTC	Jan 1 00:00:01 1972	9
63072002	-252892809	63072002	Jan 1 00:00:00 1972 UTC	Jan 1 00:00:02 1972	10
...					
78796800	-237168011	78796800	Jun 30 23:59:58 1972 UTC	Jul 1 00:00:00 1972	10
78796801	-237168010	78796801	Jun 30 23:59:59 1972 UTC	Jul 1 00:00:01 1972	10
78796802	-237168009	78796802	Jun 30 23:59:60 1972 UTC	Jul 1 00:00:02 1972	10
78796803	-237168008	78796803	Jul 1 00:00:00 1972 UTC	Jul 1 00:00:03 1972	11
...					
315964810	-1	315964810	Jan 5 23:59:59 1980 UTC	Jan 6 00:00:10 1980	19
315964811	0	315964811	Jan 6 00:00:00 1980 UTC	Jan 6 00:00:11 1980	19
315964812	1	315964812	Jan 6 00:00:01 1980 UTC	Jan 6 00:00:12 1980	19
...					
784880276	468915465	784880276	Nov 15 06:17:35 1994 UTC	Nov 15 06:17:56 1994	29
784880277	468915466	784880277	Nov 15 06:17:36 1994 UTC	Nov 15 06:17:57 1994	29
784880278	468915467	784880278	Nov 15 06:17:37 1994 UTC	Nov 15 06:17:58 1994	29
...					
911110676	595145865	911110676	Nov 15 06:17:33 1998 UTC	Nov 15 06:17:56 1998	31
911110677	595145866	911110677	Nov 15 06:17:34 1998 UTC	Nov 15 06:17:57 1998	31
911110678	595145867	911110678	Nov 15 06:17:35 1998 UTC	Nov 15 06:17:58 1998	31
...					

You can see that UTC time and the quantity returned by `gmtime()` move gradually out of synch with one another. Currently (October 1998) the two quantities have drifted 23 seconds out of sync.

The most easily used time standard today is GPS time, because GPS receivers are cheap, accurate, and available. GPS time is equal to t plus a constant. GPS was set to be zero on Jan 6 00:00:00 1980 UTC. Hence $\text{Jan 6 00:00:00 1980 UTC - Jan 1 00:00:00 1970} = 315964811\ \text{sec.}$ This is because, up to the 82 microseconds mentioned in equation (), one has

$$315964811\ \mathrm{sec} \qquad (17.0.336)$$

This number of seconds is obtained from:

$$3600\ \text{sec/hour} \times 24\ \text{hours/day} \times \quad = \quad (365\ \text{days/year} \times 8\ \text{years} + 366\ \text{days/year} \times 2\ \text{years} + 5\ \text{days}) + 11\ \text{leap seconds}$$

$$\text{Unix-C} = \mathrm{GPS} + 315964811\ \text{sec} \quad \text{for a computer started J}$$

You can see that the relationship between GPS time and Unix-C time would be

$$\text{Unix-C} = \text{GPS} + 315964800 \text{ sec} \quad \text{for a computer started Jan 1, 1980 UTC.} \tag{17.0.338}$$

if you started a perfect computer with a perfect internal clock running on Jan 1, 1970 UTC and left it running forever. On the other hand, if you started this computer off on Jan 1, 1980 its internal Unix-C time would be set to 315964800 and the relationship would be instead

$$t = 315964811 \text{ sec} + \text{TAI} - 19 = 315964792 \text{ sec} + \text{TAI}. \tag{17.0.339}$$

For a computer started at some other time, some other relationship will hold.

The final time coordinate in general use is TAI = GPS + 19 sec. This obeys the relationship

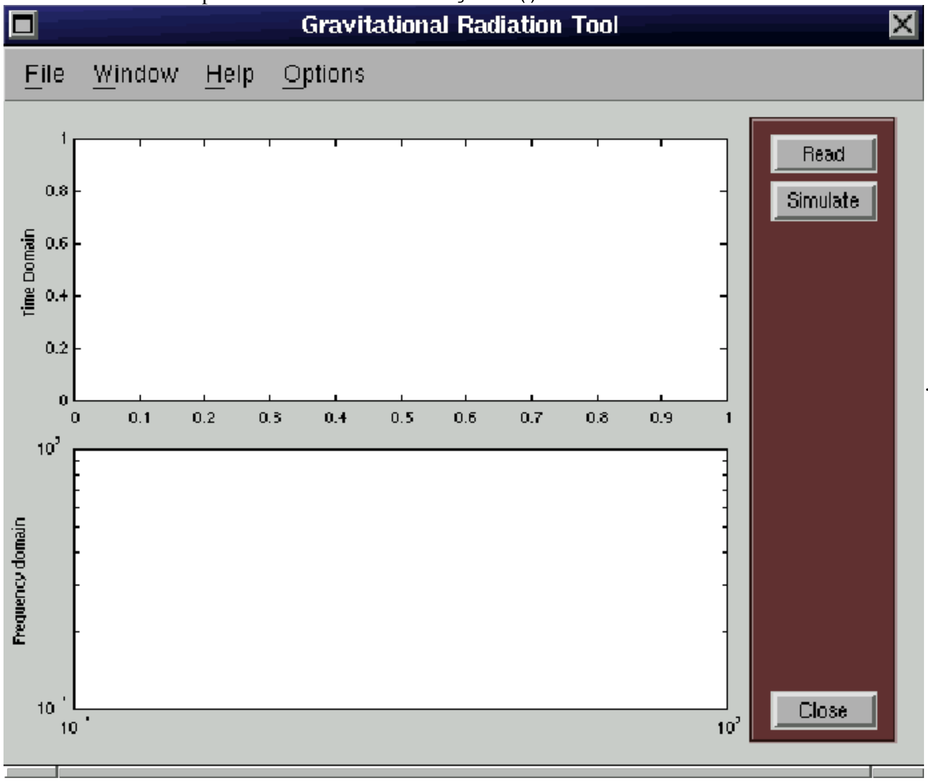
$$UTC \tag{17.0.340}$$

The GRASP library contains a pair of utility functions `utctime()` and `gpstime()` which are similar to `gmtime()`, except that they correctly generate the UTC label for Unix-C time and GPS time (respectively).

Because the number of leap seconds is not know in advance (we can not predict how the earth's rotation rate will change in the future) it is often useful to store in time records both a physical time label (for example GPS time) and in addition the number of leap seconds, in the form (TAI-UTC). *Because* the Unix-C library function `gmtime()` is broken, we can use it to print the correct ``human-readable" UTC times *almost* all of the time, by using:

$$\begin{aligned} \mathtt{gmtime}(*(\text{Unix-C} - (\text{TAI-UTC}) + 8)) &= \mathtt{gmtime}(*(\text{GPS} + 315964811 - (\text{TAI-UTC}) + 8)) \\ &= \mathtt{gmtime}(*(\text{GPS} + 315964800 - (\text{TAI-UTC}) + 19)) \\ &= * \end{aligned} \tag{17.0.341}$$

where $0, \cdots, 59$ means ``pointer to". This will work properly, except that it will return identical values for two consecutive seconds, when one occurs directly before an additional leap second is added, and the second occurs the first second of the new leap second. This is because the `gmtime()` function will never return a time of the form



XX:XX:60. The range of seconds returned by this function is .

Subsections

- [Function: `utctime()`]
- [Function: `gpstime()`]
- [Example: `testutctime`]

next up previous contents

Next: Function: utctime() **Up:** GRASP: a data analysis **Previous:** Example: trackerF **Contents**

Bruce Allen 2000-11-19