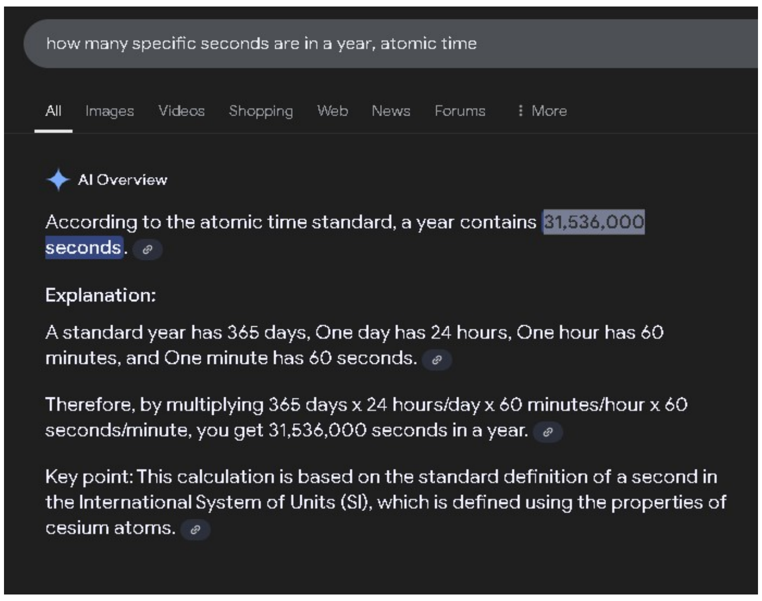
Daylight Savings Time (DST), Coordinated Universal Time (UTC), International Atomic Time (TAI), and Possible Justifications for a Universally-Accepted Model

December 17, 2024 - Tyler Scafidi



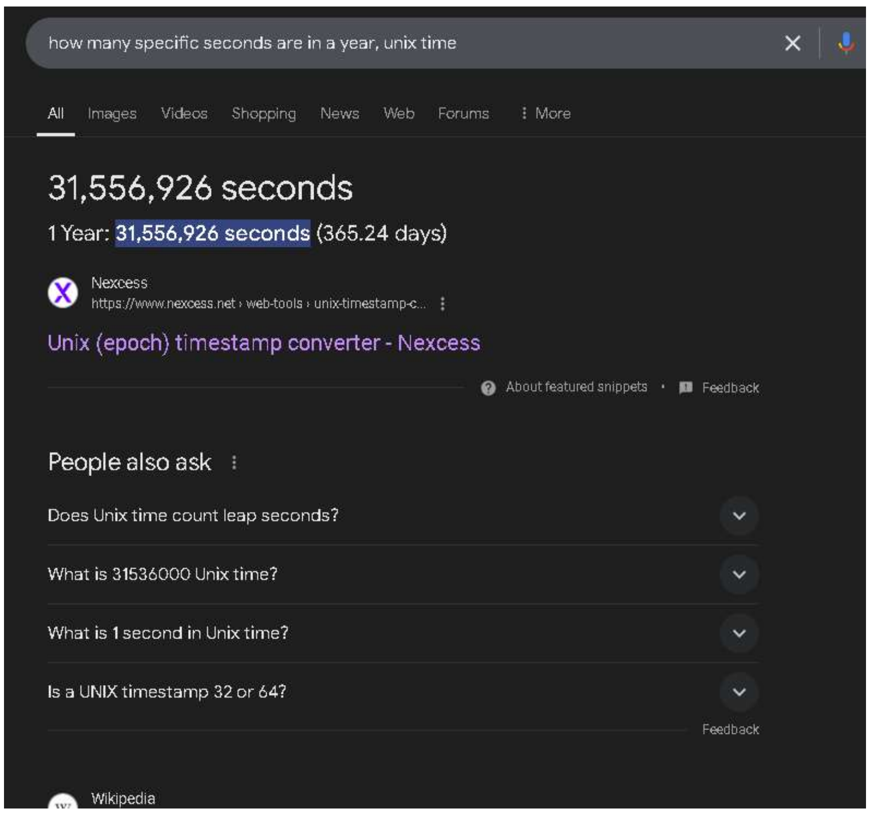
According to the atomic time standard, a year contains 31,536,000 seconds.

Explanation:

A standard year has 365 days, One day has 24 hours, One hour has 60 minutes, and One minute has 60 seconds.

Therefore, by multiplying 365 days x 24 hours/day x 60 minutes/hour x 60 seconds/minute, you get 31,536,000 seconds in a year.

Key point: This calculation is based on the standard definition of a second in the International System of Units (SI), which is defined using the properties of cesium atoms.



31,556,926 seconds

1 Year: 31,556,926 seconds (365.24 days)

Nexcess
https://www.nexcess.net › web-tools › unix-timestamp-c...

Unix (epoch) timestamp converter - Nexcess

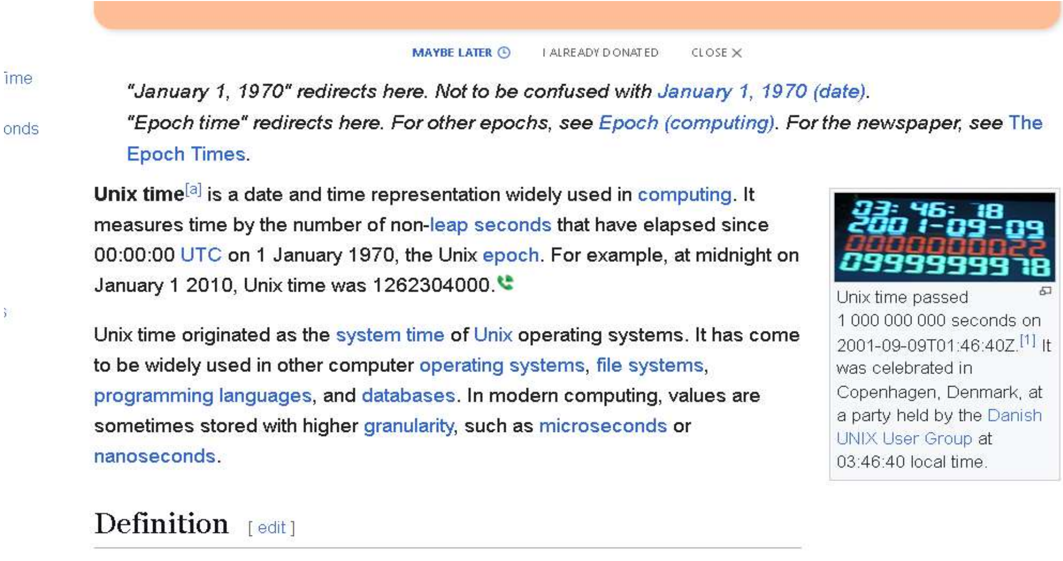
People also ask

Does Unix time count leap seconds?

What is 31536000 Unix time?

What is 1 second in Unix time?

Is a UNIX timestamp 32 or 64?



"January 1, 1970" redirects here. Not to be confused with January 1, 1970 (date).

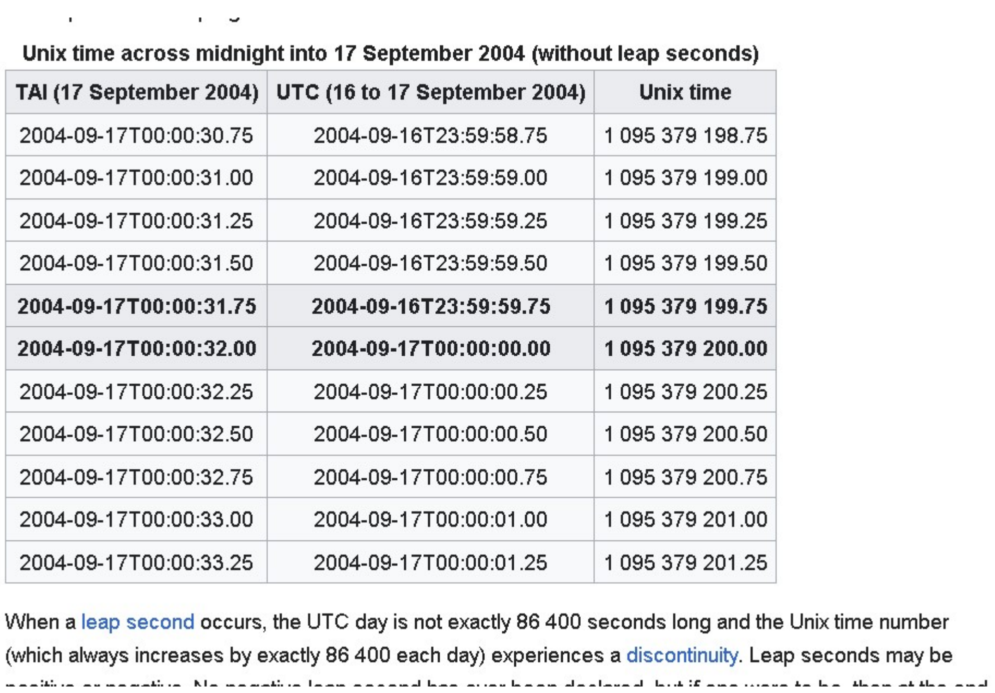
"Epoch time" redirects here. For other epochs, see Epoch (computing). For the newspaper, see The Epoch Times.

Unix time[a] is a date and time representation widely used in computing. It measures time by the number of non-leap seconds that have elapsed since 00:00:00 UTC on 1 January 1970, the Unix epoch. For example, at midnight on January 1 2010, Unix time was 1262304000.

Unix time originated as the system time of Unix operating systems. It has come to be widely used in other computer operating systems, file systems, programming languages, and databases. In modern computing, values are sometimes stored with higher granularity, such as microseconds or nanoseconds.

Unix time passed 1 000 000 000 seconds on 2001-09-09T01:46:40Z.[1] It was celebrated in Copenhagen, Denmark, at a party held by the Danish UNIX User Group at 03:46:40 local time.

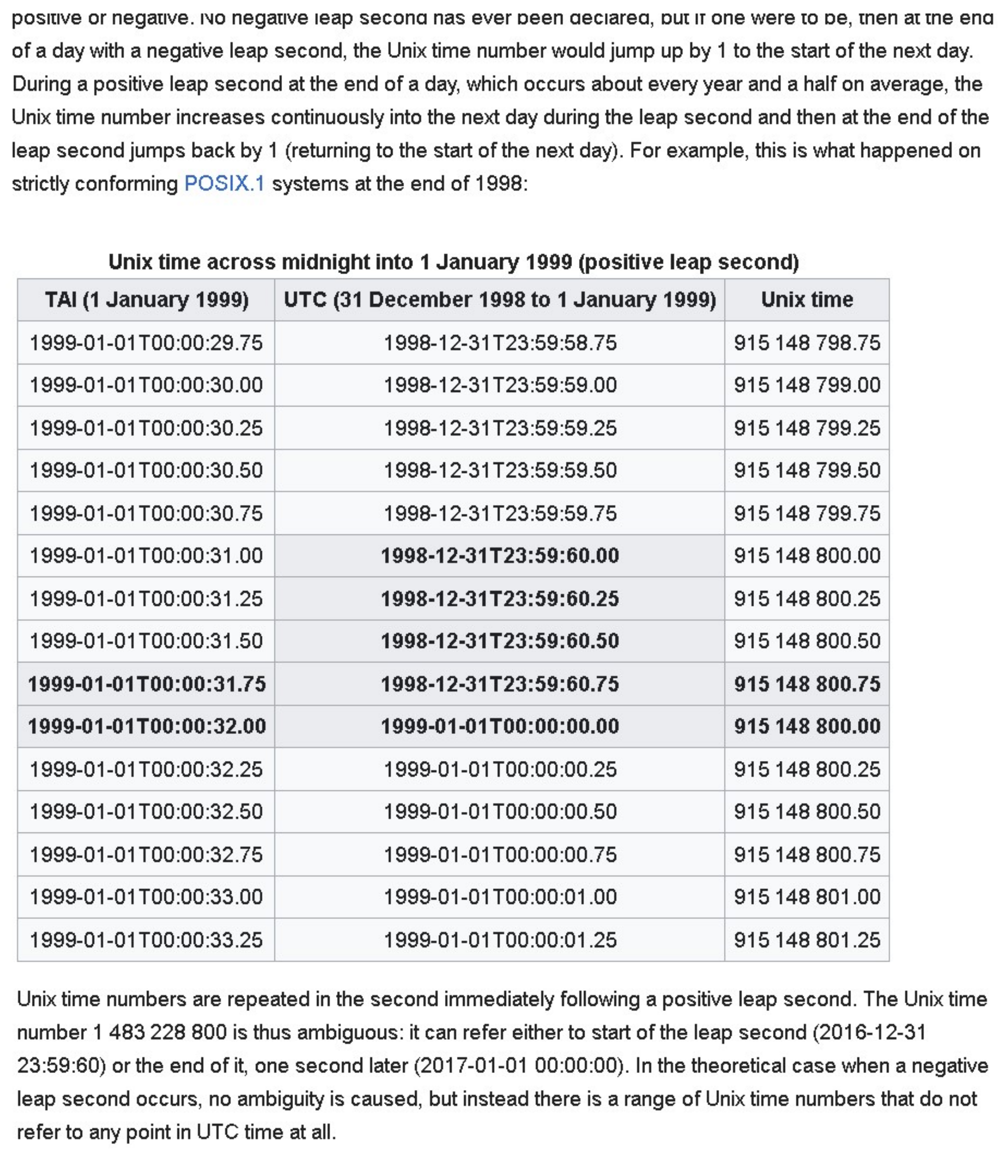
Definition [edit]



Unix time across midnight into 17 September 2004 (without leap seconds)

TAI (17 September 2004)	UTC (16 to 17 September 2004)	Unix time
2004-09-17T00:00:30.75	2004-09-16T23:59:58.75	1 095 379 198.75
2004-09-17T00:00:31.00	2004-09-16T23:59:59.00	1 095 379 199.00
2004-09-17T00:00:31.25	2004-09-16T23:59:59.25	1 095 379 199.25
2004-09-17T00:00:31.50	2004-09-16T23:59:59.50	1 095 379 199.50
2004-09-17T00:00:31.75	**2004-09-16T23:59:59.75**	**1 095 379 199.75**
2004-09-17T00:00:32.00	**2004-09-17T00:00:00.00**	**1 095 379 200.00**
2004-09-17T00:00:32.25	2004-09-17T00:00:00.25	1 095 379 200.25
2004-09-17T00:00:32.50	2004-09-17T00:00:00.50	1 095 379 200.50
2004-09-17T00:00:32.75	2004-09-17T00:00:00.75	1 095 379 200.75
2004-09-17T00:00:33.00	2004-09-17T00:00:01.00	1 095 379 201.00
2004-09-17T00:00:33.25	2004-09-17T00:00:01.25	1 095 379 201.25

When a leap second occurs, the UTC day is not exactly 86 400 seconds long and the Unix time number (which always increases by exactly 86 400 each day) experiences a discontinuity. Leap seconds may be positive or negative. No negative leap second has ever been declared, but if one were to be, then at the end

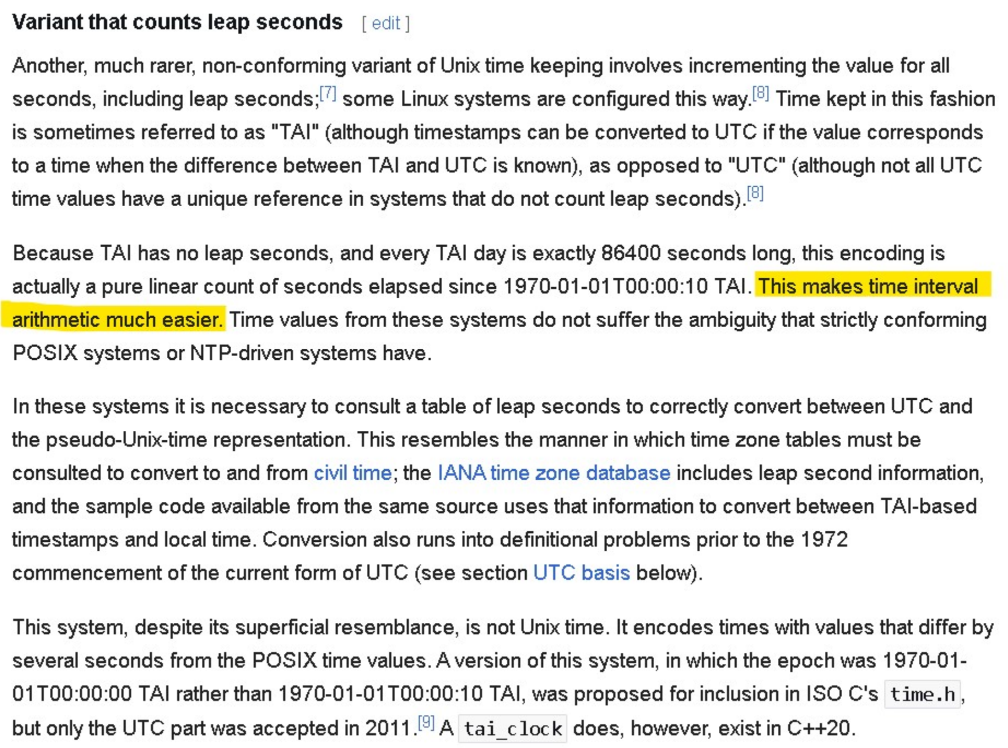


positive or negative. No negative leap second has ever been declared, but if one were to be, then at the end of a day with a negative leap second, the Unix time number would jump up by 1 to the start of the next day. During a positive leap second at the end of a day, which occurs about every year and a half on average, the Unix time number increases continuously into the next day during the leap second and then at the end of the leap second jumps back by 1 (returning to the start of the next day). For example, this is what happened on strictly conforming POSIX.1 systems at the end of 1998:

Unix time across midnight into 1 January 1999 (positive leap second)

TAI (1 January 1999)	UTC (31 December 1998 to 1 January 1999)	Unix time
1999-01-01T00:00:29.75	1998-12-31T23:59:58.75	915 148 798.75
1999-01-01T00:00:30.00	1998-12-31T23:59:59.00	915 148 799.00
1999-01-01T00:00:30.25	1998-12-31T23:59:59.25	915 148 799.25
1999-01-01T00:00:30.50	1998-12-31T23:59:59.50	915 148 799.50
1999-01-01T00:00:30.75	1998-12-31T23:59:59.75	915 148 799.75
1999-01-01T00:00:31.00	**1998-12-31T23:59:60.00**	915 148 800.00
1999-01-01T00:00:31.25	**1998-12-31T23:59:60.25**	915 148 800.25
1999-01-01T00:00:31.50	**1998-12-31T23:59:60.50**	915 148 800.50
1999-01-01T00:00:31.75	**1998-12-31T23:59:60.75**	**915 148 800.75**
1999-01-01T00:00:32.00	**1999-01-01T00:00:00.00**	**915 148 800.00**
1999-01-01T00:00:32.25	1999-01-01T00:00:00.25	915 148 800.25
1999-01-01T00:00:32.50	1999-01-01T00:00:00.50	915 148 800.50
1999-01-01T00:00:32.75	1999-01-01T00:00:00.75	915 148 800.75
1999-01-01T00:00:33.00	1999-01-01T00:00:01.00	915 148 801.00
1999-01-01T00:00:33.25	1999-01-01T00:00:01.25	915 148 801.25

Unix time numbers are repeated in the second immediately following a positive leap second. The Unix time number 1 483 228 800 is thus ambiguous: it can refer either to start of the leap second (2016-12-31 23:59:60) or the end of it, one second later (2017-01-01 00:00:00). In the theoretical case when a negative leap second occurs, no ambiguity is caused, but instead there is a range of Unix time numbers that do not refer to any point in UTC time at all.



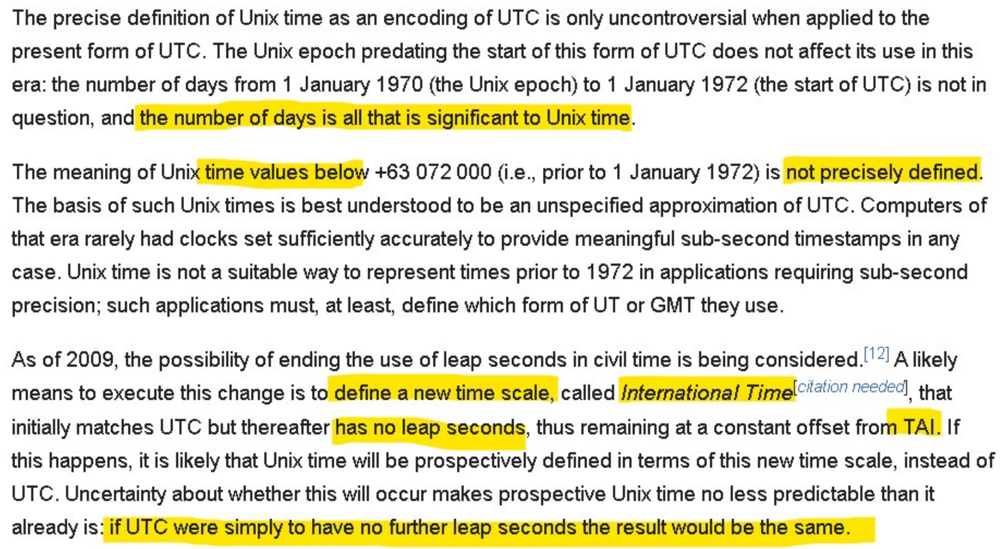
Variant that counts leap seconds [edit]

Another, much rarer, non-conforming variant of Unix time keeping involves incrementing the value for all seconds, including leap seconds;[7] some Linux systems are configured this way.[8] Time kept in this fashion is sometimes referred to as "TAI" (although timestamps can be converted to UTC if the value corresponds to a time when the difference between TAI and UTC is known), as opposed to "UTC" (although not all UTC time values have a unique reference in systems that do not count leap seconds).[8]

Because TAI has no leap seconds, and every TAI day is exactly 86400 seconds long, this encoding is actually a pure linear count of seconds elapsed since 1970-01-01T00:00:10 TAI. This makes time interval arithmetic much easier. Time values from these systems do not suffer the ambiguity that strictly conforming POSIX systems or NTP-driven systems have.

In these systems it is necessary to consult a table of leap seconds to correctly convert between UTC and the pseudo-Unix-time representation. This resembles the manner in which time zone tables must be consulted to convert to and from civil time; the IANA time zone database includes leap second information, and the sample code available from the same source uses that information to convert between TAI-based timestamps and local time. Conversion also runs into definitional problems prior to the 1972 commencement of the current form of UTC (see section UTC basis below).

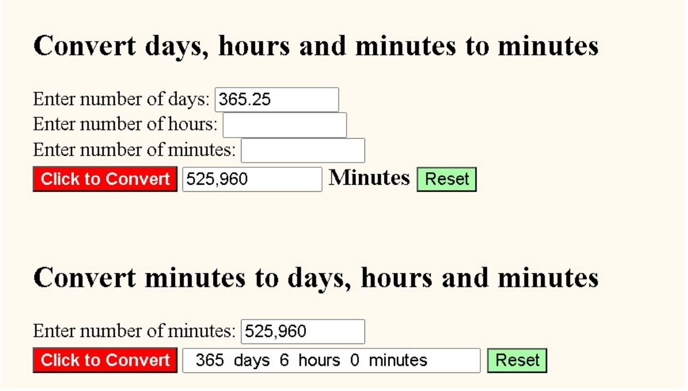
This system, despite its superficial resemblance, is not Unix time. It encodes times with values that differ by several seconds from the POSIX time values. A version of this system, in which the epoch was 1970-01-01T00:00:00 TAI rather than 1970-01-01T00:00:10 TAI, was proposed for inclusion in ISO C's `time.h`, but only the UTC part was accepted in 2011.[9] A `tai_clock` does, however, exist in C++20.



The precise definition of Unix time as an encoding of UTC is only uncontroversial when applied to the present form of UTC. The Unix epoch predating the start of this form of UTC does not affect its use in this era: the number of days from 1 January 1970 (the Unix epoch) to 1 January 1972 (the start of UTC) is not in question, and the number of days is all that is significant to Unix time.

The meaning of Unix time values below +63 072 000 (i.e., prior to 1 January 1972) is not precisely defined. The basis of such Unix times is best understood to be an unspecified approximation of UTC. Computers of that era rarely had clocks set sufficiently accurately to provide meaningful sub-second timestamps in any case. Unix time is not a suitable way to represent times prior to 1972 in applications requiring sub-second precision; such applications must, at least, define which form of UT or GMT they use.

As of 2009, the possibility of ending the use of leap seconds in civil time is being considered.[12] A likely means to execute this change is to define a new time scale, called *International Time*[citation needed], that initially matches UTC but thereafter has no leap seconds, thus remaining at a constant offset from TAI. If this happens, it is likely that Unix time will be prospectively defined in terms of this new time scale, instead of UTC. Uncertainty about whether this will occur makes prospective Unix time no less predictable than it already is: if UTC were simply to have no further leap seconds the result would be the same.

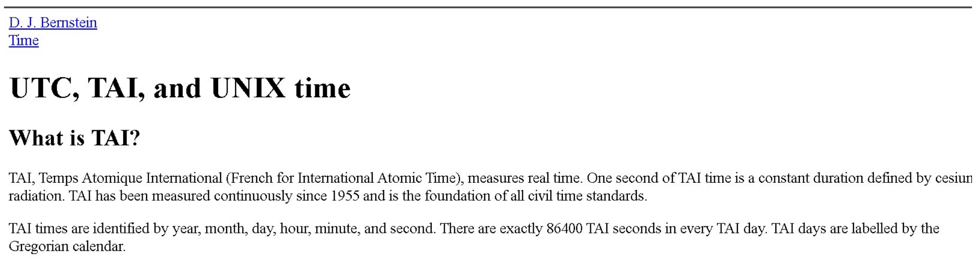


Convert days, hours and minutes to minutes

Enter number of days: 365.25
Enter number of hours:
Enter number of minutes:
Click to Convert 525,960 Minutes Reset

Convert minutes to days, hours and minutes

Enter number of minutes: 525,960
Click to Convert 365 days 6 hours 0 minutes Reset



D. J. Bernstein
Time

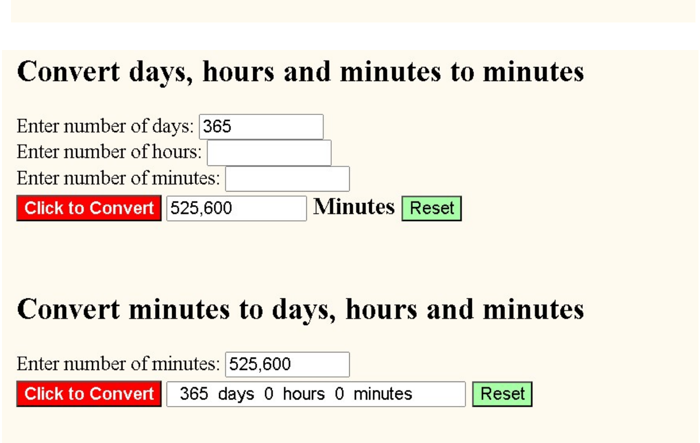
UTC, TAI, and UNIX time

What is TAI?

TAI, Temps Atomique International (French for International Atomic Time), measures real time. One second of TAI time is a constant duration defined by cesiun radiation. TAI has been measured continuously since 1955 and is the foundation of all civil time standards.

TAI times are identified by year, month, day, hour, minute, and second. There are exactly 86400 TAI seconds in every TAI day. TAI days are labelled by the Gregorian calendar.

https://cr.yp.to/proto/utctai.html



Convert days, hours and minutes to minutes

Enter number of days: 365

Enter number of hours:

Enter number of minutes:

525,600 Minutes

Convert minutes to days, hours and minutes

Enter number of minutes: 525,600

365 days 0 hours 0 minutes

Proposed Straight Time/International Atomic Time (TAI) - (a)					Average time based on 365-day annual model			
					30.41667	average days per month in a year		
SPYa	31536000	31536000			91.25	average days in a quarter		
MPYa	525600	525600			182.5	average days in six month		
HPYa	8760	8760			365	average days per year		
DPYa	365	365						
86400 seconds multiplied by 365, no leap seconds						Seconds Per Year (SPY)		
						Minutes Per Year (MPY)		
Current Time based on leap years/UTC (b)						Hours Per Year (HPY)		
						Days Per Year (DPY)		
SPYb	31557600	31557600						
MPYb	525960	525960						
HPYb	8766	8766			6	extra hours per year constant		
DPYb	365.25	365.25			12	months in a year		
					0.5	extra hours per month		
86400 seconds per day multiplied by 365.25, leap seconds					1.5	extra hours per quarter		
					3	extra hours per six months		
Hours_per_day_a			24		6	extra hours per year contstant		
Hours_per_day_b (current)			24.01644					

Where to add the extra hours? Gregorian calendar observers may benefit from 6 hours extra on their New Year's Eve.
Others: need to be considered

The time per year does not change, just how and when we acknowledge it.

The time change will not affect your seasons, only the time of day you are used to experiencing them.

If the leap year is observed every four years, a standard UTC year is 365.25 days, then December 30th of the fourth year is the last day of the year. Why they chose February 29?

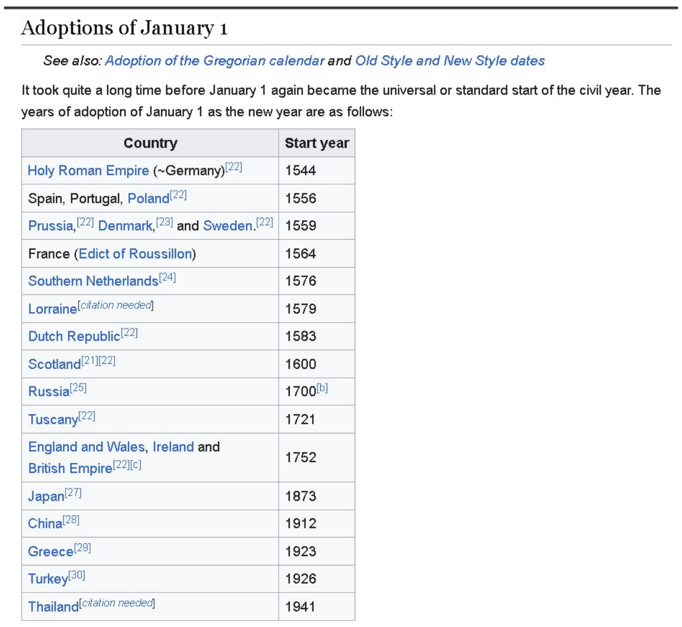
Regardless, the time does not change; we orbit at the same rate (essentially) as far as I know. There can be some deviations for drift and maybe space weather, pull by sun, and any significant disruptions to the earth that may somehow alter it's path in any way.

What kind of implication does this have on algorithmic automations, especially Finance? Is there a benefit to keeping it this way financially, less lag, or minute values that can have significant impacts?

There are some arguments that UTC time does not need or require highly-accurate time, and the clocks would be affected.

However, what is we replaced them like many of the other technologies, such as digital TV antenna requirements, modern vehicles, etc. It would need to be adopted globally.

https://en.wikipedia.org/wiki/New_Year



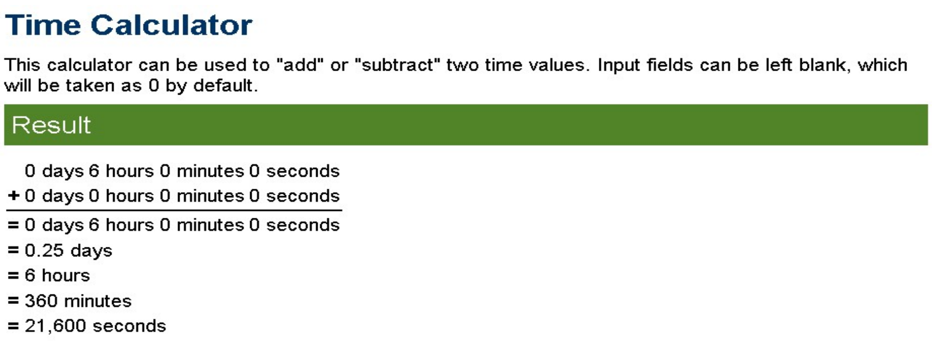
Adoptions of January 1

See also: Adoption of the Gregorian calendar and Old Style and New Style dates

It took quite a long time before January 1 again became the universal or standard start of the civil year. The years of adoption of January 1 as the new year are as follows:

Country	Start year
Holy Roman Empire (~Germany)[22]	1544
Spain, Portugal, Poland[22]	1556
Prussia,[22] Denmark,[23] and Sweden.[22]	1559
France (Edict of Roussillon)	1564
Southern Netherlands[24]	1576
Lorraine[citation needed]	1579
Dutch Republic[22]	1583
Scotland[21][22]	1600
Russia[25]	1700[b]
Tuscany[22]	1721
England and Wales, Ireland and British Empire[22][c]	1752
Japan[27]	1873
China[28]	1912
Greece[29]	1923
Turkey[30]	1926
Thailand[citation needed]	1941

https://www.calculator.net/time-calculator.html?tcday1=&tchour1=6&tcminute1=&tcsecond1=&Op=%2B&tcday2=&tchour2=0&tcminute2=&tcsecond2=&tcday3=&tchour3=&tcminute3=&tcsecond3=&ctype=1&x=Calculate



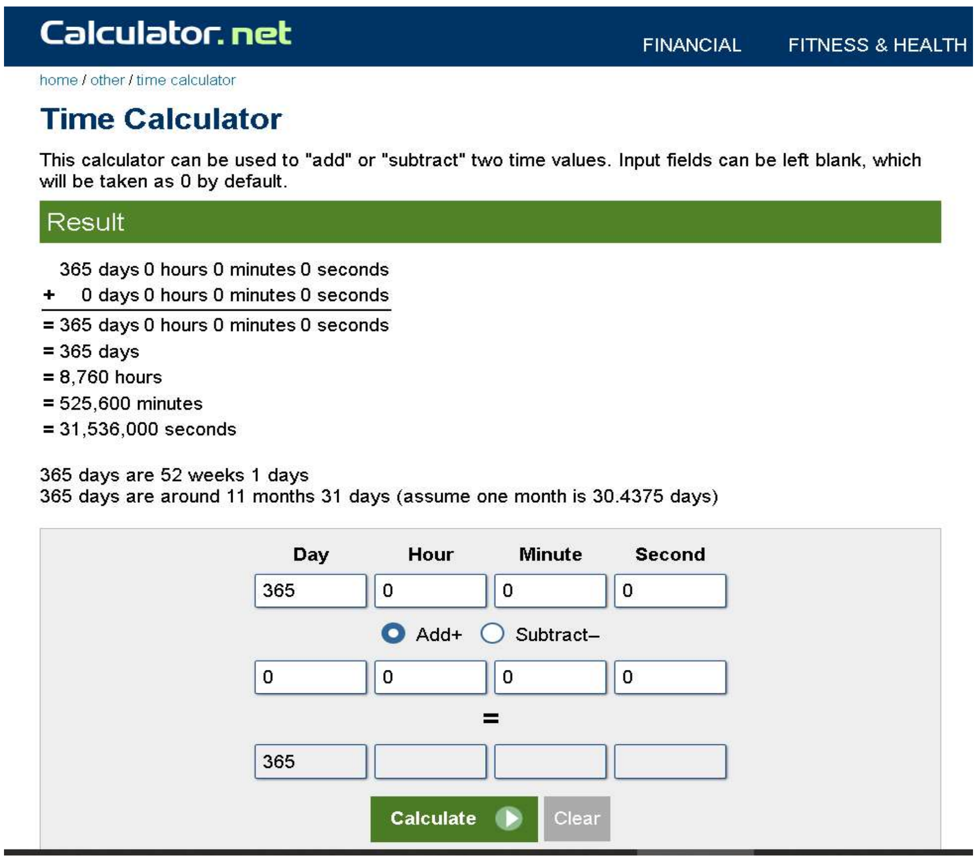
Time Calculator

This calculator can be used to "add" or "subtract" two time values. Input fields can be left blank, which will be taken as 0 by default.

Result

```
  0 days 6 hours 0 minutes 0 seconds
+ 0 days 0 hours 0 minutes 0 seconds
= 0 days 6 hours 0 minutes 0 seconds
= 0.25 days
= 6 hours
= 360 minutes
= 21,600 seconds
```

https://en.wikipedia.org/wiki/New_Year



Time Calculator

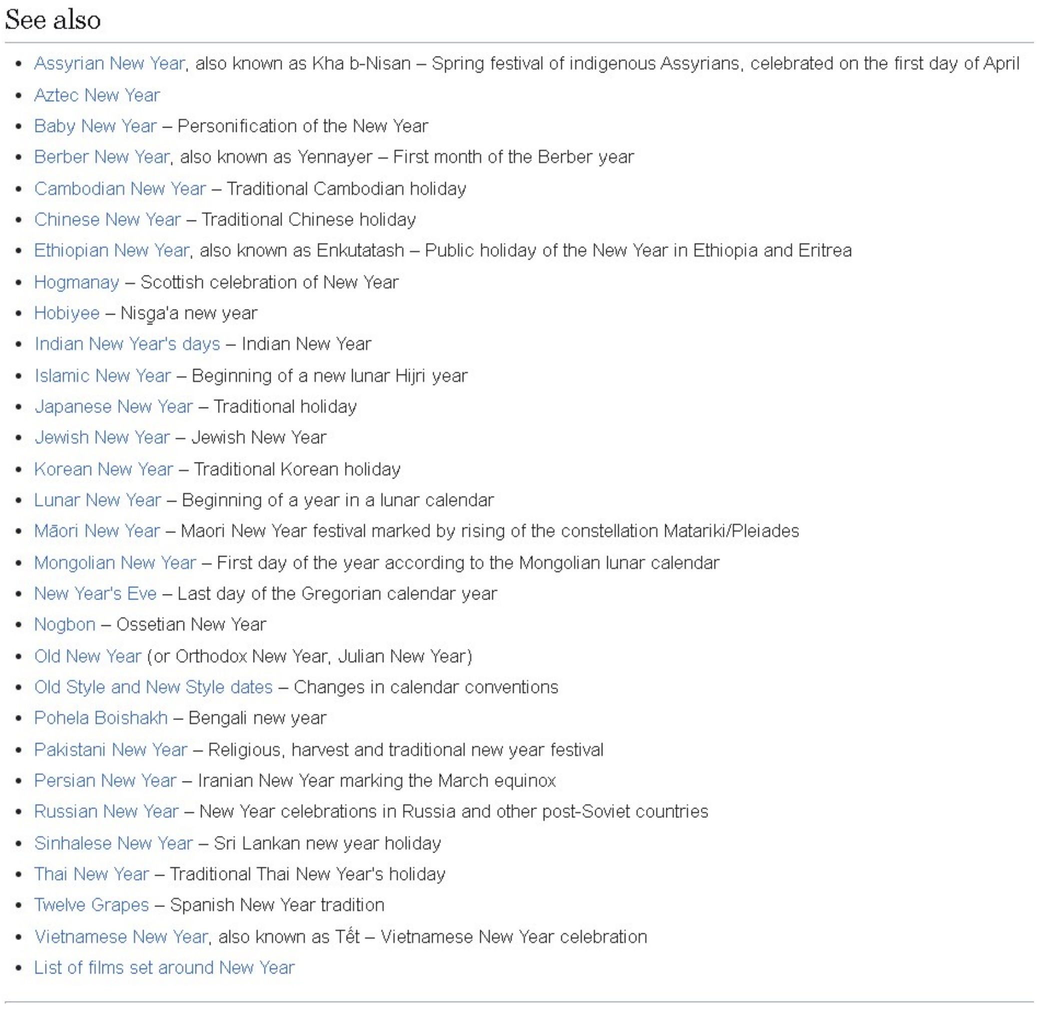
This calculator can be used to "add" or "subtract" two time values. Input fields can be left blank, which will be taken as 0 by default.

Result

```
  365 days 0 hours 0 minutes 0 seconds
+   0 days 0 hours 0 minutes 0 seconds
= 365 days 0 hours 0 minutes 0 seconds
= 365 days
= 8,760 hours
= 525,600 minutes
= 31,536,000 seconds
```

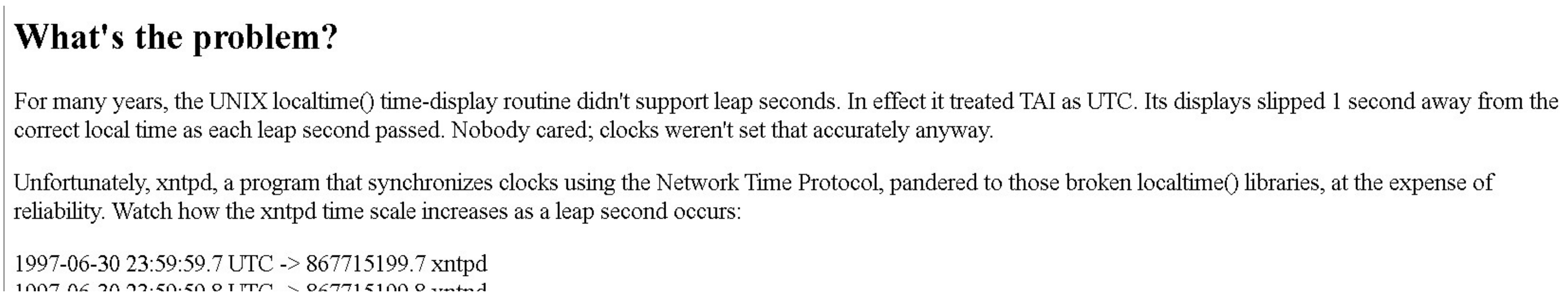
365 days are 52 weeks 1 days
365 days are around 11 months 31 days (assume one month is 30.4375 days)

Day	Hour	Minute	Second
365	0	0	0
● Add+	○ Subtract–		
0	0	0	0
=			
365			



See also

- Assyrian New Year, also known as Kha b-Nisan – Spring festival of indigenous Assyrians, celebrated on the first day of April
- Aztec New Year
- Baby New Year – Personification of the New Year
- Berber New Year, also known as Yennayer – First month of the Berber year
- Cambodian New Year – Traditional Cambodian holiday
- Chinese New Year – Traditional Chinese holiday
- Ethiopian New Year, also known as Enkutatash – Public holiday of the New Year in Ethiopia and Eritrea
- Hogmanay – Scottish celebration of New Year
- Hobiyee – Nisga'a new year
- Indian New Year's days – Indian New Year
- Islamic New Year – Beginning of a new lunar Hijri year
- Japanese New Year – Traditional holiday
- Jewish New Year – Jewish New Year
- Korean New Year – Traditional Korean holiday
- Lunar New Year – Beginning of a year in a lunar calendar
- Māori New Year – Maori New Year festival marked by rising of the constellation Matariki/Pleiades
- Mongolian New Year – First day of the year according to the Mongolian lunar calendar
- New Year's Eve – Last day of the Gregorian calendar year
- Nogbon – Ossetian New Year
- Old New Year (or Orthodox New Year, Julian New Year)
- Old Style and New Style dates – Changes in calendar conventions
- Pohela Boishakh – Bengali new year
- Pakistani New Year – Religious, harvest and traditional new year festival
- Persian New Year – Iranian New Year marking the March equinox
- Russian New Year – New Year celebrations in Russia and other post-Soviet countries
- Sinhalese New Year – Sri Lankan new year holiday
- Thai New Year – Traditional Thai New Year's holiday
- Twelve Grapes – Spanish New Year tradition
- Vietnamese New Year, also known as Tết – Vietnamese New Year celebration
- List of films set around New Year

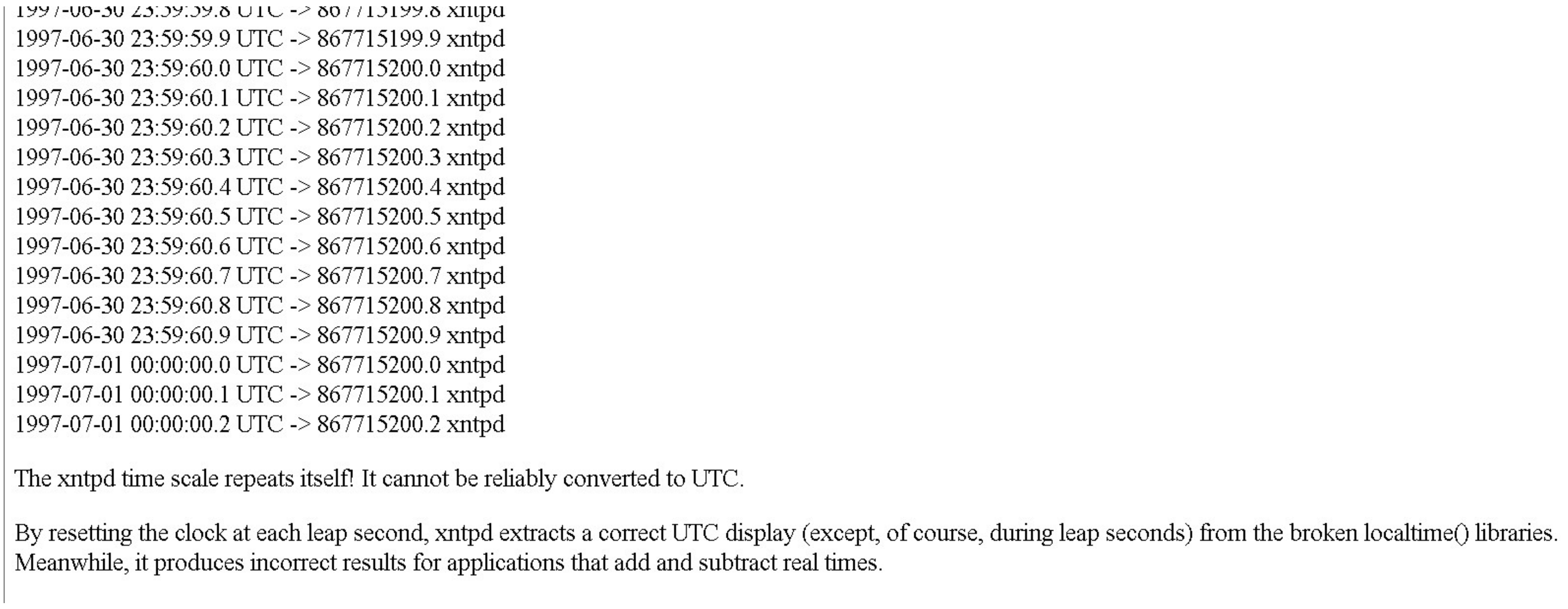


What's the problem?

For many years, the UNIX localtime() time-display routine didn't support leap seconds. In effect it treated TAI as UTC. Its displays slipped 1 second away from the correct local time as each leap second passed. Nobody cared; clocks weren't set that accurately anyway.

Unfortunately, xntpd, a program that synchronizes clocks using the Network Time Protocol, pandered to those broken localtime() libraries, at the expense of reliability. Watch how the xntpd time scale increases as a leap second occurs:

```
1997-06-30 23:59:59.7 UTC -> 867715199.7 xntpd
1997-06-30 23:59:59.8 UTC -> 867715199.8 xntpd
```

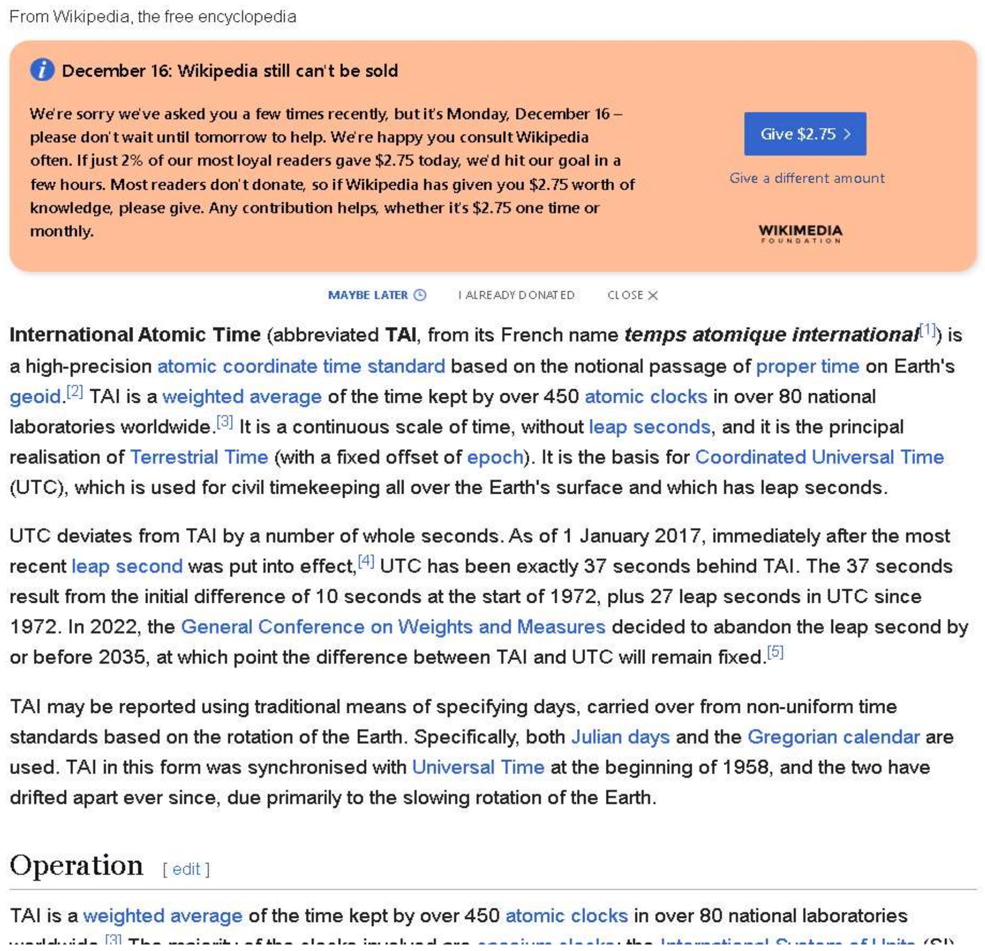


```
1997-06-30 23:59:59.8 UTC -> 867715199.8 xntpd
1997-06-30 23:59:59.9 UTC -> 867715199.9 xntpd
1997-06-30 23:59:60.0 UTC -> 867715200.0 xntpd
1997-06-30 23:59:60.1 UTC -> 867715200.1 xntpd
1997-06-30 23:59:60.2 UTC -> 867715200.2 xntpd
1997-06-30 23:59:60.3 UTC -> 867715200.3 xntpd
1997-06-30 23:59:60.4 UTC -> 867715200.4 xntpd
1997-06-30 23:59:60.5 UTC -> 867715200.5 xntpd
1997-06-30 23:59:60.6 UTC -> 867715200.6 xntpd
1997-06-30 23:59:60.7 UTC -> 867715200.7 xntpd
1997-06-30 23:59:60.8 UTC -> 867715200.8 xntpd
1997-06-30 23:59:60.9 UTC -> 867715200.9 xntpd
1997-07-01 00:00:00.0 UTC -> 867715200.0 xntpd
1997-07-01 00:00:00.1 UTC -> 867715200.1 xntpd
1997-07-01 00:00:00.2 UTC -> 867715200.2 xntpd
```

The xntpd time scale repeats itself! It cannot be reliably converted to UTC.

By resetting the clock at each leap second, xntpd extracts a correct UTC display (except, of course, during leap seconds) from the broken localtime() libraries. Meanwhile, it produces incorrect results for applications that add and subtract real times.

https://en.wikipedia.org/wiki/International_Atomic_Time



From Wikipedia, the free encyclopedia

International Atomic Time (abbreviated **TAI**, from its French name ***temps atomique international***[1]) is a high-precision atomic coordinate time standard based on the notional passage of proper time on Earth's geoid.[2] TAI is a weighted average of the time kept by over 450 atomic clocks in over 80 national laboratories worldwide.[3] It is a continuous scale of time, without leap seconds, and it is the principal realisation of Terrestrial Time (with a fixed offset of epoch). It is the basis for Coordinated Universal Time (UTC), which is used for civil timekeeping all over the Earth's surface and which has leap seconds.

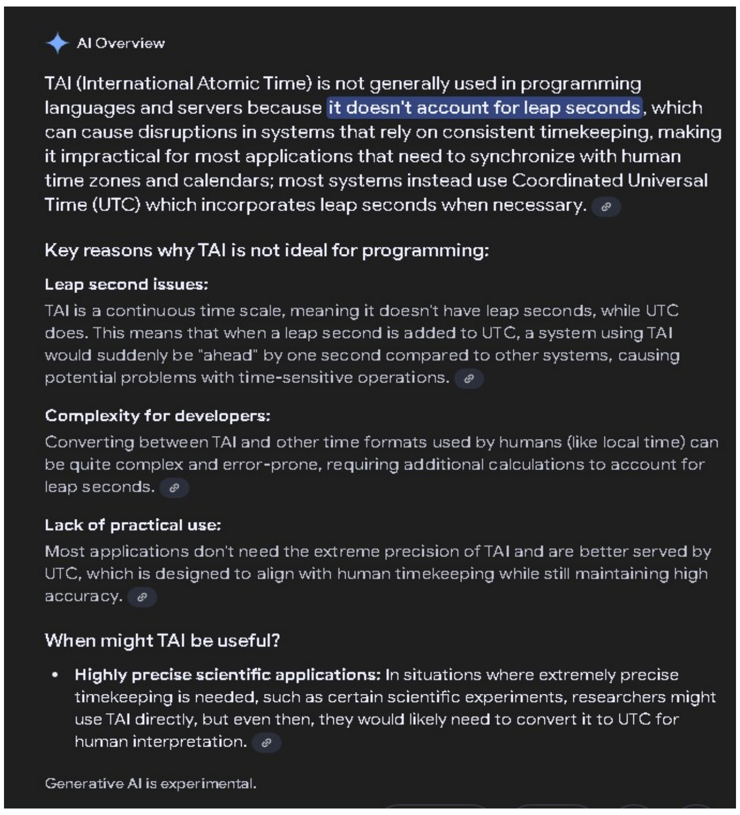
UTC deviates from TAI by a number of whole seconds. As of 1 January 2017, immediately after the most recent leap second was put into effect,[4] UTC has been exactly 37 seconds behind TAI. The 37 seconds result from the initial difference of 10 seconds at the start of 1972, plus 27 leap seconds in UTC since 1972. In 2022, the General Conference on Weights and Measures decided to abandon the leap second by or before 2035, at which point the difference between TAI and UTC will remain fixed.[5]

TAI may be reported using traditional means of specifying days, carried over from non-uniform time standards based on the rotation of the Earth. Specifically, both Julian days and the Gregorian calendar are used. TAI in this form was synchronised with Universal Time at the beginning of 1958, and the two have drifted apart ever since, due primarily to the slowing rotation of the Earth.

Operation [edit]

TAI is a weighted average of the time kept by over 450 atomic clocks in over 80 national laboratories worldwide.[3] The majority of the clocks involved are caesium clocks; the International System of Units (SI)

https://www.reddit.com/r/changemyview/comments/8v142p/cmv_computers_should_use_tai_as_time_reference/#:~:text=If%20a%20computers%20start%20using,happen%20at%20the%20same%20time.



AI Overview

TAI (International Atomic Time) is not generally used in programming languages and servers because it doesn't account for leap seconds, which can cause disruptions in systems that rely on consistent timekeeping, making it impractical for most applications that need to synchronize with human time zones and calendars; most systems instead use Coordinated Universal Time (UTC) which incorporates leap seconds when necessary.

Key reasons why TAI is not ideal for programming:

Leap second issues:

TAI is a continuous time scale, meaning it doesn't have leap seconds, while UTC does. This means that when a leap second is added to UTC, a system using TAI would suddenly be "ahead" by one second compared to other systems, causing potential problems with time-sensitive operations.

Complexity for developers:

Converting between TAI and other time formats used by humans (like local time) can be quite complex and error-prone, requiring additional calculations to account for leap seconds.

Lack of practical use:

Most applications don't need the extreme precision of TAI and are better served by UTC, which is designed to align with human timekeeping while still maintaining high accuracy.

When might TAI be useful?

- **Highly precise scientific applications:** In situations where extremely precise timekeeping is needed, such as certain scientific experiments, researchers might use TAI directly, but even then, they would likely need to convert it to UTC for human interpretation.

Generative AI is experimental.