# DISCUSSION CONCLUDING AAS 13-521

ANDREW MAIN presented on behalf of STEPHEN COLEBOURNE who was unable to attend in person. STEVE ALLEN asked to be reminded of the crossover date between UT1 and UTC-SLS for the Java® Time-Scale, to which MAIN replied November 3, 1972. ALLEN noted that there had been one leap second before that date; which MAIN affirmed; however, this date is near the start of the 'leap-seconds era' and after the 'rubber-seconds era'. UTC-SLS is only properly defined during the leap-seconds era, and there was no earlier coincidence of UT1 being equal to UTC in the leap-seconds era, because 1972 had begun with UTC already ahead of UT1. UTC-SLS is identical to UTC except during less than half an hour at the end of days which have a leap second. The criteria written out for defining the Java Time-Scale include that the scale must approximate the consensus international time scale and must match it exactly at noon each day.[*]

In the case of UTC-SLS, the second becomes stretched by one millisecond for 1000 seconds. For schemes that implement a leap second by temporarily stretching the second, GEORGE KAPLAN wondered about the implications to real-time systems. KAPLAN thought that such a scheme may cause errors in measurements of, say, rate. MAIN said that any version of UTC that has a "stretcher" to cover the leap is not the right time base for making physical measurements; that would be a case where the user would need to be aware of the difference. MARTIN BURNICKI added if one is using Unix then there are two different clock models:[†] one is the so-called *monotonic time*, which can be used to measure measurement intervals, and one is the *real-time clock* which gives the user the current UTC time.[‡]

Within the context of offering a recommendation for the ITU-R, KAPLAN asked if it was even possible to suggest the UTC-SLS approach as a possible implementation of leap seconds. MAIN replied that it was conceivable but runs into a specific problem within the purview of the ITU-R. One of the original arguments during the 'rubber-seconds era' for switching to leap seconds rather than using the frequency offsets was so the carrier wave of the radio time signal could be phase-locked to the on-time markers. Any stretching causes problems with that. If one wanted to leap by a small number of microseconds, then that would cause problems for the carrier as well. A leap precisely by an integral number of milliseconds maintains phase-lock on any carrier frequency that is a multiple of a 1000 Hz, and so on. Leaping only by seconds means that it can be phased-locked everywhere.

---

[*] http://download.java.net/jdk8/docs/api/java/time/Instant.html
http://download.java.net/jdk8/docs/api/java/time/package-summary.html

[†] IEEE Std 1003.1-2001 *Single UNIX Specification version 3*

[‡] *Editors' Note*: Following the meeting, COLEBOURNE noted for the record that the JDK8 Java Time API will have two "clock" like sources. The Clock class and *System.currentMillis*() are both defined by the Java time scale, whereas *System.nanoTime*() is intended as a monotonic source. Thus, an accurate implementation of the JDK8 specification would be able to determine the difference in time over a smoothed leap second, as *nanoTime*() would record 1001 seconds compared to *currentMillis*()/Clock recording 1000 smoothed seconds. As AAS 13-521 notes, the standard implementation of JDK8 will not be that accurate.

Assuming that Java has a clearly open choice, JIM KIESSLING asked why Java continued to work in a discontinuous time scale which presents some inconvenience, when there is effective access to a TAI transport and timing could be done with a TAI derivative. MAIN said that is a good idea if one has sufficiently easy access, and that was pretty much the agenda of his earlier talk. One should use a general timescale appropriate to one's application. KIESSLING said that as a general rule it is hard to believe that these systems do not have general access to GPS to reference TAI. Is "moving material from GPS" really beyond expectation? MAIN said that in practice the time interface does not have easy access to TAI-like time. It is a matter of "easy versus hard" but not "possible versus impossible". MAIN had played with the code that works with TAI or the equivalent and "it goes through quite a lot of effort to get hold of TAI, essentially consisting of looking at UTC which is reasonably accessible and then subtracting out leap seconds." However, that is all slightly irrelevant to the Java API, which by design uses a single compromise timescale.

ROB SEAMAN said Java is used for a lot of embedded applications. (SEAMAN noted that there was an earlier comment about Java being used in a business environment but he was unsure whether that would actually be an embedded application.) SEAMAN asked if MAIN could convey his thoughts of COLEBOURNE's paper and how the Java scale could be used in, say, a Blu-Ray Disc™ player or something else. ALLEN thought he could say something to SEAMAN's question: the Android™ telephone was the place where the Java developers discovered that the telephone's internal time was in fact GPS time and not UTC. So for such embedded applications, that is what is already happening. How exactly the Android phones hand out public UTC is somewhere else ALLEN had not explored. MAIN said that presumably they are aware of the current difference between TAI and UTC, and have warning of these things. At the moment, for general purpose platforms it is not feasible to run just on TAI or anything equivalent, because UTC is a practical requirement for user interfacing. So anything that uses TAI must be handling both.

JOHN SEAGO mentioned the subtle technicality that UTC tracked UT2 rather than UT1 in the 'rubber-seconds era', and wondered if that distinction impacts the choice of epoch for Java time. MAIN said that did not have any effect on the choice. When MAIN discussed with COLEBOURNE exactly what the definition of Java time should be in this era, the Java timescale was originally vague by definition. MAIN objected on the grounds that it did not allow the Java timescale to represent any precise data. MAIN proposed that a precise definition be used and to move the inaccuracy into the clock. So for this earlier era, some consideration was given to "which flavor of UT ought to be used." MAIN originally proposed UT2R, which was obviously not contemporaneous and would have been a proleptic usage, but it would have been a smoother timescale than raw UT1. MAIN said that COLEBOURNE reckoned that there really would not be consensus on that recommendation, and the language defining the Java timescale in terms of a 'consensus timescale' suggested that UT2R would not really qualify.

As for what UTC was tracking in the 'rubber-seconds era', MAIN explained that UTC was tracking UT2 because UTC was trying to track much more closely than it presently does, and UT2 was what had been previously used as the basis for time signals, simply because it was more regular than UT1. Presently the tracking is described in terms of UT1, but tracking in terms of UT2, or UT2R, or something along those lines is obviously more important in discerning the long-term trend. So the fact of what it is tracking is really of no relevance to the definition of the Java timescale.