Draft for approval
Published: 03/04/66

Identification

Clock Conversion Routines
Arthur Evans, Jr.

Purpose

As explained in Section BD.10.01, the hardware Calendar Clock contains an integer which is the number of micro-seconds since one micro-second after midnight on January 1, 1901, Greenwich Mean Time (GMT), and the term "Calendar Clock time" always refers to such a quantity. All times stored in the system will be Calendar Clock times, of course, but a more readable form of time must be provided for the user. Further, times supplied by the user to the system must be in a format convenient to him. This section describes the conventions and techniques used in converting Calendar Clock times to a format convenient for the user (output conversion) and in converting times produced by the user to Calendar Clock times (input conversion).

Requirements

In general, it is expected that the user will want times printed on his console to agree with the time on the clock on his wall. Since most users will be physically close to the computer, the local time _at the computer_ will usually be what is wanted. "Local time" must of course be "current local time", so the conversion routines must be cognizant of daylight saving time. The user who is far enough from the computer to be in a different time zone presumably wants times to be printed in _his_ local time, and such a user must be accommodated. Finally, it should be noted that a user giving a time to the system (for example, the time when a process is to be awakened) surely wants to give it in his own local time. Thus the input conversion routines are also concerned with this problem.

There is one final requirement -- that the user concerned with dates before 1901 or far in the future be able, if he wishes, to use the standard system time conversion routines. Although the 52-bit hardware clock will overflow on October 21, 2042, the conversion routines must handle properly times further in the future than that. They must also process properly negative calendar clock times, representing dates before 1901.

The mechanism used is to include in each process profile (i.e., user's profile -- see Section BX.0.01, The SHELL) the necessary data indicating how he wants time conversion to be done. At first glance it appears that all that is needed for this purpose is a constant to be added to the calendar time before the conversion, the constant usually being some integer (perhaps negative) times the number of micro-seconds in an hour and representing the separation between the user's time and GMT. Actually, however, things are a bit more complicated, because of daylight saving time.

The Time Conversion Table

The input and output conversion routines have available to them a table of correction factors, called the Time Conversion Table (TCT). Each "line" of this table has three entries, so that the i-th line is:

time(i) constant(i) string(i)

Whenever a time is to be converted by the output conversion routine, the time is looked up in the first column of the TCT. Let time(k) be the first time found which exceeds the argument. Then constant(k) is added to the argument before doing the conversion, and string(k) identifies the time. For example, the standard TCT at the Project MAC computer might be

24 Apr 1966 0600	-5 hours	EST
30 Oct 1966 0600	-6 hours	EDT
30 Apr 1967 0600	-5 hours	EST
29 Oct 1967 0600	-6 hours	EDT
29 Apr 1968 0600	-5 hours	EST
28 Oct 1968 0600	-6 hours	EDT
28 Apr 1969 0600	-6 hours	EST
27 Oct 1969 0600	-5 hours	EDT
(end of table)	-5 hours	EST

(The "time" entry on the last line of the table is to be filled in with the largest number which can be stored into the available field.) The times shown in column one are in GMT, and it is to be understood that the TCT in the computer will contain the corresponding calendar clock time. Column two will be a signed integer which is the equivalent number of micro-seconds. The dates given represent the last Sunday in April and in October, and the times given are the times when daylight saving time goes in and out. (0600 GMT is 0100 EST.)

The table shown provides that all times before 24 April 1966 or after 27 October 1969 will be printed as EST, and that times between these two dates will be printed correctly as EST or EDT. As the system ages, the table will be extended so that, at any given instant, all times within the next (say) three years will be printed correctly. If it seems desirable to keep the table length constant, lines can simultaneously be deleted from the top of the table.

Note the third column of the table. This string is printed with each time (and is, indeed, part of the character string which is the time) to make unambiguous the meaning of the printed time.

Now consider the user in, say, Los Angeles. In his user profile there will be a similar table with different entries, reflecting these differences:

1. Los Angeles is -8 hours from Greenwich, not -5 hours, so that the magnitudes of the times in the second column will (in general) be increased by three hours.

2. Daylight saving time starts and stops on different dates in California and Massachusetts. The dates in the first column must be altered accordingly.

3. The strings in column three will be "PST" and "PDT".

Clearly, an appropriate table could be constructed for any place in the world -- even one like Afghanistan which is +4 hours 26 minutes from Greenwich.

Now consider the man who usually uses the computer at MAC, but who happens to be using it from, say, Phoenix for a few days. The first time he logs in from Phoenix, he may well find it useful to have a constant two hours added to each column two entry.

Now consider the user who habitually travels around the world using the system. He may choose to get all times in GMT, so his TCT would contain the single entry

(end of table)	+0 hours	GMT

As above, "end of table" represents the largest integer which can be stored in the avaliable field.

Finally, consider the user who is concerned with times before 1901 and/or after the upper limit of the (hardware) Calendar Clock. Clearly, such a user will need input and output conversion routines of greater sophistication than those needed by most users. For example, going before 1901 or after 2099 requires knowing that neither 1900 nor 2100 is a leap year; going before 1572 requires knowing about the Gregorian and Julian Calendars; and going more than 1965 years into the past requires knowing about AD and BC. On entry, the conversion routines check with a quick test that the data is within the "standard" range. If so, all is well. If not, a more sophisticated routine is called to do the conversion. Although this latter routine is part of the supervisor, it will only be fetched into core when (and if) it is actually needed. Thus no one "pays for" this routine unless he uses it.

One final point should be noted: The conversion routines are all on the user's side of the outer security wall, so he is free to replace them by routines of his own if he so chooses.

The Output Conversion Process

We now consider in more detail the process of converting a calendar time to a form interesting to a user. Actually, the output conversion is done in two steps, only the first of which will be discussed here. Given a calendar time, the output conversion routine will produce for the caller the following eight values:

YEAR	An integer, four decimal digits precision, which is the calendar year.
MONTH	A two digit integer from one to twelve which is the month number.
DAY	A two digit number from one to 31 which is the day of the month. (DAY will never be greater than the number of days in the month in question.)
HOUR	A two digit integer from zero to 23 which is the number of hours since midnight.
MIN	A two digit integer from zero to 59 which is the number of minutes since the hour.
SEC	A two digit integer from zero to 59 which is the number of seconds since the minute.
USEC	A six digit integer from zero to 999999 which is the number of micro-seconds since the second.
ZONE	A three character string which identifies the time zone, such as "EDT".

It should be clear that all of the information in the calendar clock time is reflected in these eight quantities, and that the calendar time could be recreated from them. Further, it is clear that it is easy to write a routine which, given these eight quantities, will produce a character string such as

```
1323.2 EST  13 Jan 1966
```

or any equivalent string that seems desirable.

The Time Zone Table

The routine that does input conversion of times must have available to it a table of time zone abbreviations. The user may type "MST" as part of a time in order to make unambiguous what he means, so the input conversion routine must know that the string "MST" means -7 hours from GMT. The Time Zone Table (TZT) has two entries on each line: a three-character string and a signed integer which is the number of micro-seconds from GMT. A standard table is available to the system containing the time zone abbreviations most used at the installation, but the user may supply in his profile a TZT tailored to his own use.

The process of processing a time zone abbreviation supplied by the user takes place as follows: If the user has his own TCT, the abbreviation is first looked up in the third column, the assumption being that he is unlikely to input an abbreviation unless he has provision to output it. If the abbreviation is found, the column 2 entry on that line is used as the correction constant; but if it is not found, further searching takes place. If the user has supplied his own TZT, that is searched. If not, though, the system standard TZT is used.

The Input Conversion Process

A time supplied by the user to the system falls into one of two cases: Either a time zone abbreviation is explicitly given or one is not. In either case, though, the first part of the input conversion process is the same -- the typed time/date is converted to a binary integer as if it were GMT. If the user has supplied a time zone abbreviation (such as "EDT"), it is looked up as described above. The appropriate correction is then subtracted from the binary integer to get the proper calendar time.

If a time zone abbreviation is not supplied, the binary integer is looked up in the usual way in the TCT and the corresponding constant is subtracted from (not added to) it. If there is no ambiguity (see below), the result is the desired calendar time.

There are several anomalies or ambiguities that may be detected in the input conversion process, a few of which are as follows:

1. The time zone supplied by the user may be incompatible with the date and time typed. While it may be reasonable to accept "EST" in June, it is almost surely improper to refer to "EDT" in December.

2. The time 0130 does not exist on the last Sunday in April (in Massachusetts), since at 0100 EST on that date the legal time jumps to 0200 EDT. Presumably a reference to 0130 EST means a time 31 minutes later than 0059 EST, but a reference to 0130 with no qualification is probably wrong and a reference to 0130 EDT is surely wrong.

3. The time 0130 is ambiguous on the last Sunday in October, since there are two of them. (At 0200 EDT the time becomes 0100 EST.) Of course, either 0130 EST or 0130 EDT is unambiguous and acceptable.

4. The data may be poor. The time "1168" (where the last two digits are the number of minutes after the hour) or the time "2505" are probably wrong, as is the 32-nd of the month, the 30-th of February or month number 13.

5. Abbreviated date-times may be permitted. For example, the user may omit typing the year if he is referring to "this year", or the date if it is "today". However, there are cases where the intended meaning is clear to the user, but not so clear to the program. (For example, at two minutes past midnight, does "2358" refer to today or to yesterday? Similarly, on January 5, does December 28 refer to last year or to this year?)

The processing of problems such as the above is dependent on the current setting of the "no questions" switch in the process profile. Unless the user has indicated no questions, he will be given an error (or warning) message and asked to correct the data. Otherwise, the conversion routine will make a default interpretation of the data and go on. In absentee-user processes the problem need never arise if the user makes a practice of always specifying times completely.