Published: 05/03/67

Identification

Time Conversion
calendar_output, calendar_input
L. B. Ratcliff

Purpose

The calendar_output procedure performs the basic computations required to convert internal calendar clock time to more conventional external forms involving year, month, day, etc. calendar_input computes an internal calendar clock time when given a time in parameter form, i.e., year, month, day, etc.

This procedure is based on conventions and techniques discussed in section BD.10.02.

Discussion

The internal form of a clock time is a signed 71-bit integer. This time is in microseconds relative to 0000 hours GMT, January 1, 1901. The equivalent external form consists of the components:

year (AD)	integer	from 0001 to 9999
month	integer	1 to 12
day of month	integer	1 to 31
hour	integer	0 to 23
minutes	integer	0 to 59
seconds	integer	0 to 59
microseconds	integer	0 to 999999
time zone	char.string	3 characters
day of week	integer	1 to 7 (1 = Sunday)

Output conversion (calendar_output) produces all these components; input conversion (calendar_input) requires all but day of week.

While the calendar procedures perform the basic time conversion and are available to any user, with their lengthy argument lists they will not be called directly by most users. The procedures get_calendar and put_calendar are provided to call calendar_output or calendar_input and format the

time. (See Section BY.15.03.) It is anticipated that other special formats, in addition to those provided by get_calendar and put_calendar, will be provided by procedures which call calendar_output, calendar_input, get_calendar and put_calendar.

Implementation of calendar_output and time_conversion_table is planned for Phase II. Implementation of calendar_input and time_zone_table is planned for Phase IV.

Time_Conversion_Table

Time_conversion_table is an external data base which is aware only of Eastern Standard Time (EST) and Eastern Daylight Time (EDT). This data base is subject to modification by the user who wishes to know or specify times in other zones. The primary function of time_conversion_table is to accomodate time changes within a specific time zone. Each entry in the standard table specifies a time at which EST or EDT becomes effective. Each table entry contains three items; the initial version contains the following entries:

time(i)			constant(i)	string(i)
30 Apr	1967	0600	-5 hours	EST
29 Oct	1967	0600	-4 hours	EDT
29 Apr	1968	0600	-5 hours	EST
28 Oct	1968	0600	-4 hours	EDT
2 Jan	10000	0000	-5 hours	EST

The time and constant entries are values in microseconds. Note that the time(i) are the times at which time changes take place and are in order with earlier times first. The last entry must contain a time greater than or equal to 2 Jan 10000 0000 to indicate the end of the table.

For output conversion, the items time(i) determine the range in which the specified time occurs. The corresponding constant(i) expresses the difference between GMT and time in the zone identified by string(i). If the specified time is between time(k-1) and time(k), the corresponding conversion values are constant(k) and string(k). For input conversion, the specified time zone is compared to the items string(i) to determine a corresponding constant(i) required to compute an internal item (GMT). A detailed discussion of the time conversion table appears in Section BD.10.02.

Time_Zone_Table

If the specified time zone is not found in time_conversion_table during input conversion, an auxiliary table, time_zone_table, is searched. Each entry in time_zone_table consists of two items - constant and string. The constant(i) are values expressing the time difference in hours between GMT and the zones indicated by the string(i). Standard entries are:

constant(i)	string(i)
-5	EST
-4	EDT
-6	CST
-5	CDT
0	GMT
-8	PST
-7	PDT
-7	MST
-6	MDT
0	<SP> <SP> <SP>

The ordering is from most likely to least likely. Each user may replace this table with his own version if he wishes to input time in zones other than his own which do not appear in the standard time_zone_table. The last entry must contain the string <SP> <SP> <SP>. The time_zone_table must not contain more than 50 entries.

Usage: calendar

To obtain the external components of an internal clock time:

```
call calendar_output (clock,year,month,day,hour,min,
sec,musec,zone,weekday);
```

To convert external components into an internal clock time:

```
call calendar_input (clock,year,month,day,hour,min,
sec,musec,zone);
```

Arguments are declared as follows:

```
dcl clock fixed bin (71), zone char (3),
          (year,month,day,hour,min,sec,weekday) fixed
           bin (17), musec fixed bin (35);
```

Implementation

The calendar procedures are aware of the Gregorian calendar system only. Users who are interested in ancient systems, reform calendars, or other concurrent calendars will require a more sophisticated procedure whose first function will be to determine the required calendar system and then to call the appropriate computational procedure, such as calendar_output. One rather arbitrary check is made by calendar_output to ascertain that the computed Gregorian year is in the range 1 through 9999. If the year is outside that range then an error condition is signalled. Beyond that, the calling procedure may establish the time at which the Gregorian calendar is effective. (Pope Gregory's calendar was not universally accepted in 1582. It was first used by England in 1752, by several other countries in 1900, and yet others well into the 20th century.)

The following discussions describe the implementations of calendar_output and calendar_input. Expressions involve the PL/I generic function mod (a, b) to express a(modulo b) and the notation [x] to indicate the integral part of x.

A. calendar_output

Clock time is a 71-bit integer. Its value is in microseconds relative to 0000 January 1, 1901 GMT. The return values and clock time are declared as follows:

```
dcl (year,month,day,hour,min,sec,weekday)
          fixed bin (17), musec fixed bin (35),
          zone char (3), clock fixed bin (71);
```

The return values are determined by the following steps:

1. Using clock and time_conversion_table, compute time in the specified time zone. The value of clock is compared with the entries time(i). If time(k) is the first entry which exceeds clock then

```
local_time = clock+constant(k)
zone = string(k)
```

2. Separate local_time into an integral number of days (ndays) and fractional part of a day (rday).

   ndays = [local_time/8.64e10]

   rday = mod (local_time, 8.64e10)

3. Using rday (the number of microseconds since 0000 of the day) compute hour, min, sec, and musec.

4. Compute number of days (d111) relative to Monday, January 1,1 (Gregorian calendar system).

   d111 = ndays + 693960

   where 693960 is the number of days from January 1,1 to January 1, 1901. If d111 is negative or exceeds 3652058 (Dec. 31,9999) then seterr (BY.11.01) is called to record the error and condition (calendar_output_err) is signalled.

5. Determine weekday. Sunday = 1, Monday = 2, etc.

   weekday = mod (d111+1,7)+1

6. Compute the year and day of the year. If the date is between January 1, 1901 and February 28,2100 inclusive, a part of the ensuing computation can be bypassed. Thus if $0 \leq$ ndays ≤ 72742 by pretending every year is a leap year (that is, adjusting ndays to the value, fake, that it would have if every year were a leap year), the actual year and day_of_year can be found. Noting that there are 1461 days in four years (including one leap year), 3 days must be added to ndays for every full 4-year period in ndays, and 1 day must be added for each additional 365-day year, with an adjustment (subtracting one day) if the actual day is December 31 of a leap year.

```
fake = ndays + 3[ndays/1461]
       + [mod(ndays,1461)/365]
       - [mod(ndays,1461)/1460]

year = [fake/366] + 1901
day_of_year = mod (fake,366)
Go to step 8.
```

7. Additional calendar adjustments are required to compute the year if it is outside the range checked in step 6. Again pretending every year is leap year, and noting that 400 years of Gregorian Time contain 146097 days and that 100 year periods ending in a centesimal year not divisible by 400 contain 36524 days, the proper adjustments can be made. Observe that in a 400 year period (e.g. 1601 through 2000) 303 years contain 365 days and that in a 100 year period described above there are only 24 leap years. Thus it is necessary to add 303 days for each 400 year period, 76 days for each additional 100 years, 3 days for each additional 4 year period, and 1 day for each additional year with an adjustment for the last day of a 400 year or 4 year period. The values x, y and z are computed first to simplify the computation.

```
x = mod (d111, 146097)

y = mod (x, 36524)

z = mod (y, 1461)

fake = d111 + 303 [d111/146097]
       + 76[x/36524] + 3[y/1461]
       + [z/365] - [x/146096] - [z/1460]

year = [fake/366] + 1

day_of_year = mod (fake, 366)
```

8. Using day_of_year (which is the number of days past January 1 of the year), compute month and day. First, set j = 1 if year is a leap year. Set j = 0 if year is not a leap year. Now, by giving every month 31 days, month and day can be obtained. Calculate,

   b = day_of_year - 59 -j

   to determine whether or not month falls beyond February. If b is positive make the adjustment

   day_of_year = day_of_year + 3 - j + 2[b/153] + [mod(b,153)/61]

   This adds 3-j days for February, 2 days for each full 5-month period beyond February and 1 day for each full 2-month period beyond the last full 5-month period, taking advantage of the pattern of numbers of days in the months: 31,28/29,
   31,30,31,30,31,
   31,30,31,30,31.

   In any case

   month = 1 + [day_of_year/31]

   day = 1 + mod (day_of_year,31)

9. Return.

B. calendar_input

   All components have been specified except, possibly, time zone. First, a composite 71-bit representation of the time, T, is created. The following input arguments are used:

   y = year - 1
   m = month
   d = day
   j = 1 if year is a leap year, 0 if not.

The number of days (ndm) in the m-1 full months is the value of the mth element of the array (0,31,59, 90,120,151,181,212,243,273,304,334), plus j when m>2. Then the total number of days is:

$$ndays = 365y + [y/4] - [y/100] + [y/400] + ndm + (d-1) - 693960$$

where 693960 is the number of days from Jan. 1,1 to Jan. 1,1901.

$$T = (((ndays*24+\underline{hour})*60+\underline{min})*60+\underline{sec})*10^{6}\underline{musec}$$

The final step from the composite time, T to an internal time (GMT) involves subtraction of the constant for the specified time zone from T. However, the following problems must be taken into account.

1. The caller may implicitly specify the time zone in his time_conversion_table (which contains information to handle time changes within only one zone) by using <SP> <SP> <SP> as the input character string for zone. If the table contains only one entry, there are no complications. However, for multiple-entry tables, the internal clock time is determined by using the constant

   $$K = \max (constant(i))$$

   for all i.

   Each entry of time_conversion_table is checked until

   $$T < time(i) + K$$

   then the corresponding constant(i) is subtracted from T to obtain the internal clock time. The value of K is assumed to be the constant corresponding to a daylight time entry. Its use here establishes the convention that standard time stays in effect until daylight time becomes meaningful. Note that in this case 0130 April 30, 1967 is accepted as standard time and identical to 1230 April 30, 1967 which is interpreted as daylight time, as is 0130 October 29, 1967.

2. The caller may explicitly specify the time zone which is in his time_conversion_table. Here, let

   N = min (constant(i))

   for all i. The table is searched until

   T < time(i) + constant(i)

   If constant(i) = N then the internal clock time is T - constant(i). However, if constant(i) ≠ N and time zone is not string(i) the search is continued. This allows standard time to be accepted any time since the next entry in a properly constructed time_conversion_table is a "standard time" entry and contains constant(i) = N. (Note that if T is a time in July, 1967 and zone = EST then T < time(2) + constant(2), constant(2) ≠ N, and zone ≠ string(2). However, T <time(3) + constant(3) and constant(3) = N.) If time zone is string(i), one additional test is required based on the assumption the current entry indicates a transition from daylight time to standard time. The time is specified to be daylight time. It is meaningful except during the first hour of this interval. If

   time(i-1)≤T-constant(i-1)<time(i-1)+constant(i)-constant(i-1)

   an error is signaled since the explicit time does not exist. Also, if the (i-1)th entry does not exist an error is signaled; the initial entry should be for standard time.

3. The caller may explicitly specify a time zone which is in time_zone_table. No error rests are made; T - constant(i) is the internal clock time.

The four errors in calendar_input are handled by calling seterr (BY.11.01) to record the error, then signalling calendar_input_err. The errors and their codes are:

ci_001 The explicit time given does not exist.

ci_002 Time_conversion_table is improperly constructed.

ci_003 Specified time is beyond the range of time_conversion_table.

ci_004 Error in time_zone_table. Maximum number of entries allowed is 50. The last entry must contain the string <SP> <SP> <SP>.