To: MTB Distribution

From: Gary C. Dixon

Subject: New calendar_clock active function

Date: July 15, 1975

INTRODUCTION

Several people have informally reported problems in using the existing date/time active functions for writing exec_coms. These problems center around the process of comparing two dates within an exec_com. This is a special problem in system administration exec_coms and absin files, which must take special actions depending upon the date/time of a previous action or expected occurrence. Usually such date/time values are returned by the system active function, or are stored in a value segment by a previous invocation of the exec_com (absin file).

THE PROBLEM

There is no active function which returns a complete date and time in a form which allows ascending date/times to be sorted or compared in ASCII collating sequence. The closest approximation to such a date/time string is the date_time active function, which returns:

"mm/dd/yy hhmm.m zzz dow"

An exec_com using this form to compare two date/times must first check the year before comparing the remainder of the date, and its time comparisons are limited to the tenths-of-a-minute returned by this form. It also has additional problems: two times which are identical except for the time zone will not sort/compare properly; and the return value is difficult to store in a value segment because the spaces between items force the value to be enclosed in quotes and the command processor strips off these quotes after each use. These problems make the handling of dates within exec_coms unnecessarily complex. This MTB proposes a new calendar_clock active function which will provide the date and time in a form which does not have the problems mentioned above.

PROPOSED SOLUTION

calendar_clock would return a 34 character date/time string in the following form:

yy.mm.dd__hh:mm:ss.ssssss_zzz__dow

All items begin at a fixed position within the string and are padded out to a fixed size so that the substr active function can be used to extract particular items of interest. In addition, items are separated from one another by underscore's (_) rather than spaces so that the return value can be treated as a single character string. (1)

The date is in a new format which can be sorted/compared. It gives the year-of-century, month number, and day number, all separated by periods. (2) The time is expressed in hours, minutes, seconds, and micro-seconds. convert_date_to_binary_ already accepts the hour:minute:second format, but would have to be modified to accept fractional seconds used in the above format. this extended-accuracy time format. A time zone is included to qualify the interpretation of the hours in a time value (more on this below). The string ends with a three character abbreviation for the day of the week.

PROBLEMS WITH THE TIME ZONE

calendar_clock solves all of the problems mentioned above, except for the time zone problem: two date/time values returned by calendar_clock which are identical except for their time zones are not sorted/compared properly. There are two possible solutions to the problem:

1. One solution is to rank each of the time zone identifiers, assign increasing numbers to each rank, and output the numbers in the date/time value in addition to the time zone identifier. While this would cause the date/time values to be sorted/compared properly, having a ranking number in the output is not very meaningful to the user; furthermore, convert_date_to_binary_ would have to be modified to accept such ranking numbers, but the existence of such numbers would make the syntax of the date/time language ambiguous.

(1) The use of underscores in place of spaces in convert_date_to_binary_ date/time strings was approved in MCR 1119.

(2) This new date format was also approved in the MCR 1119 convert_date_to_binary_.

2. A second solution is to ignore the problem. It will usually not be necessary to compare two times given for different time zones, and comparison of times within a single zone (standard time vs. daylight time) occurs properly.

3. The third solution is to add a control argument to all of the date/time active functions (including calendar_clock) which selects the time zone in which the output value is to be expressed. This would give the user the option of having all of the time values to be compared expressed in a specified zone, other that the current time zone. This might be done by adding a -zone control argument to each of the date/time active functions. For example:

```
calendar_clock -zone est
```

I recommend the third solution, as being a compatible change which offers the greatest flexibility to the user.