Measuring Days and Hours: An Introduction to the Handy Uses of the IN, END, LAG, FIRST and LAST statements.
Catherine Loveless Schmitt, Bureau of the Census, Washington, DC

ABSTRACT
Using a simplified example from a real ACCESS data base that contains incomplete date and time information, this paper describes several helpful features in SAS that can be used in the DATA step. The DATA step processing has some handy features and the review of the SAS Log can provide very helpful information. Some comparisons using simple procedures will also be included.

INTRODUCTION
There are several features in SAS that can provide an introductory analyst with useful information quickly. This discussion will use a simple data set in order to quickly discuss the following helpful features in the data step for the beginning SAS user:

- **IN=** flags used in set and merge statements
- **END=** flags.
- **_N_** = 1 to test for first observation in a DATA step.
- **FIRST** and **LAST** processing used with SET and MERGE statements.
- **LAG** function

**TABLE 1**

<table>
<thead>
<tr>
<th>Txtdate</th>
<th>Txthour</th>
<th>NumberReqs</th>
<th>Numberhits</th>
</tr>
</thead>
<tbody>
<tr>
<td>01/APR/2005</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01/APR/2005</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01/APR/2005</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01/APR/2005</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01/APR/2005</td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01/APR/2005</td>
<td>13</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01/APR/2005</td>
<td>14</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01/APR/2005</td>
<td>15</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01/APR/2005</td>
<td>16</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Note that the date and hour are stored in character formats. These will be converted to numeric variables
within the SAS program for use in numeric DO LOOPS.

```sas
data backup;
  SET msbusy.bs_backup;
  ndate=input(txtdate,date11.);
  nhour=txthour*1;
  drop txtdate txthour;
```

**DATE SEQUENCING:**
SAS stores date values in a numeric format that is calculated from the duration of days since 01Jan1960. This is important for sequencing the data, and it will be helpful when looking for gaps in the date sequence. For this application it will be important to find the earliest and latest date in a sequence. First, the data will need to be sorted by (earliest) date. Since the program is sorting the data, it will be more efficient to sort by (earliest) hour within each date at the same time.

```sas
Proc sort data=backup;
  By ndate nhour;
```

In order to find the very earliest date, simply select the first observation (if `_N_ = 1`)
The last observation, which has the latest date and time will be found using the END= option. In this application,
Two new data sets FIRST and LAST were created. FIRST has a single variable called STARTDATE. LAST has a single variable named ENDDATE.

**EXAMPLE 1:**

```sas
data first(keep=ndate rename=(ndate=startdate))
  last(keep=ndate rename=(ndate=enddate));
set backup end=end1;
  if _n_ = 1 then output first;
  * executes when the first observation is processed;
if end1 then output last;
  *the end flag is set to 1 when the last observation has been read;
```

Selections from the SAS LOG:
```
NOTE: There were 23853 observations read from the data set WORK.BACKUP.
NOTE: The data set WORK.FIRST has 1 observations and 1 variables.
NOTE: The data set WORK.LAST has 1 observations and 1 variables.
```

END= is used to select the last record in the data set.
Note the use of KEEP= and RENAME= options used at the definition of the data sets.
NDATE is renamed STARTDATE for the earliest date and ENDDATE for the latest date.

**DUPLICATE CHECKING**
Since it is time consuming to pass through the data, an error checking statement to search for duplicates can be added to execute at the same time. Since the data has been sorted by date and time, a SET statement with a BY option demonstrates the helpful uses of the FIRST. and LAST. options within the DATA step.
It is important that there are no duplicate hours in a day and the use of FIRST. And LAST. flags will check for that. The flags FIRST. and LAST. are automatically created when data is SET or MERGED with a By statement. The flag FIRST.NHOUR will be set to 1 when the first occurrence of an hour is found. Since this sort was performed by date(NDATE) then hour(NHOUR) the flag will be set to one when the first occurrence of hour within date is found. The flag LAST.NHOUR will be set to 1 when the last occurrence of an hour(NHOUR) within a date(NDATE) is found. The following Table displays an example of how the FIRST. And LAST. processing works.

### EXAMPLE 2 DATA:

<table>
<thead>
<tr>
<th>obs</th>
<th>NDATE</th>
<th>NHOUR</th>
<th>FIRST.</th>
<th>LAST.</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>(date7. format)</td>
<td></td>
<td>NHOUR</td>
<td>NHOUR</td>
</tr>
<tr>
<td><em>N</em> = 1 01Jan05</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td><em>N</em> = 2 02Jan05</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td><em>N</em> = 3 02Jan05</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td><em>N</em> = 4 02Jan05</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td><em>N</em> = 5 02Jan05</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td><em>N</em> = 6 02Jan05</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td><em>N</em> = 7 02Jan05</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

**EXAMPLE 2**

```sas
DATA first(keep=ndate
rename=(ndate=startdate))
last(keep=ndate
rename=(ndate=enddate))
error;

set backup end=end1;
by ndate nhour;

* this line checks for duplicates;
if not(first.nhour and last.nhour)
then output error;
if _n_ = 1 then output first;
if end1 then output last;
run;
```

**SELECTIONS FROM THE SAS LOG**

NOTE: There were 23853 observations read from the data set WORK.BACKUP.
NOTE: The data set WORK.FIRST has 1 observations and 1 variables.
NOTE: The data set WORK.ERROR has 0 observations and 8 variables.
NOTE: The data set WORK.LAST has 1 observations and 1 variables.

The two conditions FIRST.NHOUR and LAST.NHOUR should always be true. If only one is true, then there is an error and the hour is not unique for that date within the data set. The statement IF NOT(FIRST.NHOUR AND LAST.NHOUR) is very useful for testing for duplicates.
COMPLETING INCOMPLETE DATA

A complete list of dates with 24 hours per date is created in the following data set called CHECKLIST. This “dummy” data set is constructed in order to compare the number of observations, in the original data base BACKUP with the expected number of observations in CHECKLIST. Remember, the program already checked for duplicates.

The SAS processor will step through at least once, by the use of DO UNTIL loop. A single variable(XIPS) is added with the value -1.

COMPLETING DATA - EXAMPLE 3

```
data checklist;
  *this step creates 24 hours of records;
  * for each date in the system;
  merge first last ;

  *The data set first contains the variable startdate;
  *The data set last contains the variable enddate;
  ndate=startdate;

  do until (ndate > enddate);
    do nhour=0 to 23;
      xips=-1;
      output ;
    * Note the explicit use of The output statement;
      end; * hour;
    ndate=ndate+1;
    end; * date;
  run;
```

USING LOGIC FOR DIAGNOSTIC AND ERROR CHECKING

Here is a very useful feature for diagnostics and error checking. The original data set BACKUP is match merged to CHECKLIST, by NDATE (date) and NHOUR(hour). The MERGE includes the IN= option for logic checking:

- If the data came from CHECKLIST and BACKUP then all is okay.
- If the data came from BACKUP but did not find a match in CHECKLIST, that would be an ERROR.
- If the data came from CHECKLIST but did not find a match in BACKUP then dummy values for the variables need to be created in the COMPLETE data set; AND the data is OUTPUT to the MISSINGDATA data set in order to determine how many observations were missing.
EXAMPLE 4

```sas
data error missingdata complete;
  format ndate date9.;
  merge backup(in=b1)
    checklist(in=c1);
  by ndate nhour;
    if not(c1) then output error;
    if not(b1) then do;
      numberIps=-1;
      numberReqs=-1;
    output missingdata;
    end;
  if c1 then output complete;
run;
```

Selection from the SAS log:

NOTE: There were 23853 observations read from the data set WORK.BACKUP.
NOTE: There were 24072 observations read from the data set WORK.CHECKLIST.
NOTE: The data set WORK.ERROR has 0 observations and 8 variables.
NOTE: The data set WORK.MISSINGDATA has 219 observations and 8 variables.
NOTE: The data set WORK.COMPLETE has 24072 observations and 8 variables.

The IN= is an automatic flag that can used in a SET or MERGE statement to determine if the data came from one of the data sets referenced in the MERGE, (or alternatively SET) statement. The use of the IN= is useful as an error checking device, in the following way. All records in the data set CHECKLIST should match to a record in the data set BACKUP, so the condition NOT(C1) should always be false. If there are missing records, and it is supposed that there are, then the condition would be NOT(B1).

Review how the logical conditions are used below to check for missing data and create new records with dummy values in order to create a complete data set. In the above example, the condition B1 will be true when the data exists in data set BACKUP. If a record has a value 1 for the flag NOT(C1) then the resulting record did not originate from CHECKLIST, which would be an error since CHECKLIST was created to be a complete data set from earliest to last date, with full 24 hours per day.

Missing records would be found by use of the statement: IF NOT (B1). IF a record has a value 1 for the flag NOT(B1) then that data is missing from the BACKUP data set and a dummy record need to be produced and output to the data set COMPLETE. The COMPLETE data set is created by the use of the IN(C1) statement.

CHECK THE LOG

In order to check that the correct number of observations were created, I created a data step that calculated the number of days and multiplied the number of days by 24 to get the expected number of observations. I also added this part of the program to demonstrate how to use the PUT _ALL_ statement in order to read information directly from the LOG file.

EXAMPLE 5

```sas
data expected_obs;
  merge first last;
  days= (enddate-startdate+1);
  obs= days*24;
  put all ; run;
```

```sas
startdate=16437 enddate=17439 days=1003 obs=24072
_ERROR_=0 _N_=1
NOTE: There were 1 observations read from the data set WORK.FIRST.
NOTE: There were 1 observations read from the data set WORK.LAST.
NOTE: The data set WORK.EXPECTED_OBS has 1 observations and 4 variables.
```
INCOMPLETE DATA
Incomplete data can be checked using the original data set BACKUP.

```sas
proc freq data=backup;
    format ndate date9.;
    tables ndate/list out=freq_date nopercent
    nocum   noprint; run;
```

Since the data was sorted from earliest to latest date, the LAG function can be used to check for consecutive days (NDATE) and hours (NHOURS).

**EXAMPLE 6**

```sas
data missingdays(keep=ldate mdate ndate ndate2) missinghours;
    format ndate ldate 8.;
    set freq_date end=end1;
    ndate2=ndate;
    ldate=lag(ndate);
    if (_n_ > 1) then do;
      if ldate+1 < ndate then do;
        mdate=ldate;
        put _all_;
        do until (mdate = (ndate-1));
          mdate=mdate+1;
          output missingdays;
        end;
        end;
      end;
    if count < 24 then output missinghours ;
    run;
```

MISSING HOURS
Here is a simple way to process the original data set BACKUP to check for days with less than 24 hours of data.

**EXAMPLE 7**

```sas
proc print
    data=freq_date;
    where count < 24;
    run;
```

Selection from the SAS log
```
note=16447 ldate=16442 end1=0
count=24 percent=0.1006162747
ndate2=16447 mdate=16442 _ERROR_=0
_n_=7
ndate=16937 ldate=16932 end1=0
count=24 percent=0.1006162747
ndate2=16937 mdate=16932 _ERROR_=0
_n_=493
Note: There were 995 observations read from the data set work.freq_date.
Note: The data set missingdays has 8 observations and 4 variables.
Note: The data set missinghours has 26 observations and 6 variables.
```

Selection from the SAS log
```
Note: There were 26 observations read from the data set work.freq_date.
WHERE count<24;
```
A PROC PRINT with a WHERE clause will print output data set FREQ_DATE. The value COUNT will display how many hours of data are missing for each partially incomplete day of data. The original data set BACKUP had 18 days with more than 1 hour of data but less than 24 hours of information for that day.

**INCOMPLETE DAYS (ALL or PARTLY MISSING)**

For a list of incomplete days, use the following SAS procedures, as an alternative to example 6 (that uses the LAG() function):

**EXAMPLE 8**

```sas
proc freq data=complete;
  where numberips=-1;
  format ndate date9.;
  tables ndate/list
  out=incomplete_days
  nopercent nocum noprint;
```

Selection from SAS log

```
NOTE: There were 259 observations read from the data set WORK.COMPLETE.
WHERE numberips=-1;
NOTE: The data set WORK.INCOMPLETE_DAYS has 34 observations and 3 variables.
```

The number of hours that have missing data in data set COMPLETE are 259. The number of incomplete days, (all missing or partly missing) are 34 days.

The following SAS code will find the number of days that were missing SOME hours of data from the data set COMPLETE.

**EXAMPLE 8A**

```sas
proc print data=incomplete_days;
  where count < 24;
  title "Incomplete days of data";
  run;
```

Selection from the SAS log

```
NOTE: There were 26 observations read from the data set WORK.INCOMPLETE_DAYS.
WHERE count<24;
```

The following SAS code will find the number of days that were missing ALL hours of data from the data set COMPLETE.

**EXAMPLE 8B**

```sas
proc print data=incomplete_days;
  where count = 24;
  title "Missing days of data";
  run;
```

Selection from the SAS log

```
NOTE: There were 8 observations read from the data set WORK.INCOMPLETE_DAYS.
WHERE count=24;
```

COUNT is a tally of the number of times within one day(NDATE) that data was missing (-1). If the COUNT = 24 then all hours(NHOUR) of data were missing for that day(NDATE).

For those keeping track, data set MISSINGDAYS (example 6) had 8 observations. Thus the two values from example 6 and example 8B match.

**Alternative Method: PROC SUMMARY**

The following SAS code will find number of days where all values of hourly data (NHOUR) were set to missing. These were the days that did not exist in data set BACKUP.
EXAMPLE 9

```sas
proc summary data=complete;
  by ndate;
  var numberhits;
  output out=sum_days sum=;
data miss_date;
  set sum_days;
  if numberhits=-24;
proc print data=miss_date;
title "Missing days of data";
run;
```

NOTE: The data set WORK.MISS_DATE has 8 observations and 4 variables.

If all values for NUMBERHITS for each hour(NHOUR) of the day(NDATE) are = -1 they would sum to -24.

And finally, an display of those eight (8) days that did not exist in data set BACKUP, but were added to data set COMPLETE.

SAS OUTPUT

<table>
<thead>
<tr>
<th>Obs</th>
<th>ndate</th>
<th><em>TYPE</em></th>
<th><em>FREQ</em></th>
<th>Numberhits</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>07JAN2005</td>
<td>0</td>
<td>24</td>
<td>-24</td>
</tr>
<tr>
<td>2</td>
<td>08JAN2005</td>
<td>0</td>
<td>24</td>
<td>-24</td>
</tr>
<tr>
<td>3</td>
<td>09JAN2005</td>
<td>0</td>
<td>24</td>
<td>-24</td>
</tr>
<tr>
<td>4</td>
<td>10JAN2005</td>
<td>0</td>
<td>24</td>
<td>-24</td>
</tr>
<tr>
<td>5</td>
<td>12MAY2006</td>
<td>0</td>
<td>24</td>
<td>-24</td>
</tr>
<tr>
<td>6</td>
<td>13MAY2006</td>
<td>0</td>
<td>24</td>
<td>-24</td>
</tr>
<tr>
<td>7</td>
<td>14MAY2006</td>
<td>0</td>
<td>24</td>
<td>-24</td>
</tr>
<tr>
<td>8</td>
<td>15MAY2006</td>
<td>0</td>
<td>24</td>
<td>-24</td>
</tr>
</tbody>
</table>

CONCLUSION

This paper has a dual focus. First, there is a useful introduction to some important SAS statements that can be used in the DATA step. These five DATA step options (IN, END, LAG() function, FIRST. And LAST. are very powerful. That was the intention of demonstrating their usefulness in a real application. The SAS statements used in combination can check for duplication, errors, incompleteness and missing data, as demonstrated above. Finally, simple PROCedures were run for comparison.

ACKNOWLEDGEMENTS

I would like to thank Beverly Fransen and Lisa Pyle for their helpful comments in developing this topic.

CONTACT INFORMATION

I enjoyed writing this paper and I look forward to your comments. You may contact the author at:

Catherine Loveless Schmitt