Paper 5035-2020

History Carried Forward, Future Carried Back: Mixing Time Series of Differing Frequencies

Mark Keintz, Wharton Research Data Service

ABSTRACT

Many programming tasks require merging time series of varying frequency. For instance, you might have three data sets (YEAR, QTR, and MONTH) of data, each with eponymous frequency and sorted by common ID and date variables. Producing a monthly file with the most recent quarterly and yearly data is a hierarchical last-observation-carried-forward (LOCF) task. Or, you might have three irregular times series (ADMISSIONS, SERVICES, TESTRESULTS), in which you want to capture the latest data from each source at every date encountered (event-based LOCF). These are tasks often left poorly optimized by most SQL-based languages, in which row order is ignored in the interests of optimizing table manipulation.

This presentation shows how to use conditional SET statements in the SAS® DATA step to update specific portions of the program data vector (that is, the YEAR variables or the QTR variables) to carry forward low frequency data to multiple subsequent high frequency records. A similar approach works just as well for carrying forward data from irregular time series. We also show how to use "sentinel variables" as a means of controlling the maximum time-span data is carried forward; that is, how to remove historical data that has become "stale." Finally, we demonstrate how to modify these techniques to carry future observations backward (NOCB[1]), without re-sorting data.

INTRODUCTION

The principal objective of this paper is to show that a single well-designed DATA step can be much more efficient at LOCF tasks than the usual proc sql or multiple data step approaches. It takes advantage of a specific underlying property of the SET statement – namely that variables read by a SET are not automatically reset to missing in every iteration of the DATA step. Instead those variables are "retained", i.e. they keep the most recently retrieved values until that SET statement is executed again, or any SET statement reading the same variables.[2] The examples below will show how conditionally reading from each input data set will trivially carry forward values from low frequency series (e.g. YEAR) while processing higher frequency series (e.g. QTR or MONTH). The programming for this task is easily expanded to mixing more than two series.

SAMPLE DATA

Consider two data sets, YEAR and QTR, each sorted by ID and DATE, as in tables 1 and 2. The annual dates are for fiscal years (id XX ends its fiscal year in June, and YY ends its fiscal year in December), and the quarterly dates are the end of fiscal quarters.

ID	DATE	AstA ($mm)	LbtA ($mm)
XX	30JUN2014	821	281

[1] "NOCB" will be used to represent "Next Observation Carried Back".
[2] This is also a property of the MERGE and UPDATE statements, but discussion here will focus on using the SET statement.

ID	DATE	AstA ($mm)	LbtA ($mm)
XX	30JUN2015	799	303
XX	30JUN2016	804	322
YY	31DEC2014	1,401	904
YY	31DEC2015	1,427	950
YY	31DEC2016	2,550	962

Table 1: Sample Fiscal **YEAR Data (2 ID's, 3 Years)**

ID	DATE	SalQ ($mm)	EmpQ (1,000)
XX	30JUN2014	132.3	13.5
XX	30SEP2014	128.9	12.6
XX	31DEC2014	138.3	13.5
XX	31MAR2015	112.0	11.3
XX	30JUN2015	115.5	11.7
XX	31SEP2015	140.2	14.2
:::	:::	:::	:::
XX	31MAR2017	98.9	9.9

Table 2: Sample Fiscal QTR Data (Showing part of 1st ID Only)

The LOCF goal is to produce a table with annual data carried forward for each matching or trailing quarter. In Table 3 below the values in the *yellow cells with bold italics* are carried forward from a single annual record through the subsequent quarterly records. Table 3 also has YR_DATE, the DATE value taken from the YEAR data set. This is useful for indicating the "vintage" of the annual data.

ID	DATE	YR_DATE	AstA	LbtA	SalQ ($mm)	EmpQ (1,000)
XX	30JUN2014	30JUN2014	821	281	132.3	13.5
XX	30SEP2014	***30JUN2014***	***821***	***281***	128.9	12.6
XX	31DEC2014	***30JUN2014***	***821***	***281***	138.3	13.5
XX	31MAR2015	***30JUN2014***	***821***	***281***	112.0	11.3
XX	30JUN2015	30JUN2015	799	303	115.5	11.7
XX	31SEP2015	***30JUN2015***	***799***	***303***	140.2	14.2
:::	:::	:::			:::	:::
XX	31MAR2017	***30JUN2016***	***804***	***322***	98.9	9.9

Table 3: Yearly Data Carried Forward Into Quarterly Series (Part of id XX Only)

THE USUAL SOLUTIONS: PROC SQL OR MULTIPLE DATA STEPS

Typically LOCF is handled either through SQL code or multiple data steps. Each can do the task, but each has deficiencies.

THE PROC SQL APPROACH

Here's a likely PROC SQL solution:

```
proc sql noprint;
  create table yrqtr as
  select *
    I
  left join
    year (rename=(date=yr_date))
  on year.id=qtr.id
  and yr_date<=date<intnx('year',yr_date,1,'s')
  order by id,date;
quit;
```

It's a left join of QTR with YEAR, so the new table will have at least one record per quarter. For each ID the quarterly date cannot precede the annual date nor follow it by a year or more. The code is relatively simple, but has two primary disadvantages:

1. Complexity. It quickly becomes noticeably more complex if there are three (or more) time series to mix (e.g. YEAR, QTR, and MONTH). Yes, conceptually it would be just a matter of nesting sub-queries (i.e. left join the MONTH with the subquery defined as the left join of QTR with YEAR above). But the complexity becomes much more difficult when mixing irregular series (event-based LOCF).
2. Lack of Efficiency. Even if there are no sub-queries, PROC SQL doesn't take full advantage of the fact that the data are sorted. In particular the
   `and yr_date<=date<intnx('year',yr_date,1,'s')`
   constraint means proc sql will do a Cartesian comparison of dates for all records with the same ID. If a given ID had 10 YEAR records and 40 QTR records, that would mean 400 date comparisons – a time-consuming process. Adding a 120 MONTH series would add another 40*120=4800 date comparisons per id.

THE MULTIPLE DATA STEP APPROACH

Why a Single Step Merge Won't Work

At first you might expect that multiple data steps are not needed, and that a standard match-merge (MERGE plus BY statements) would solve the problem as in:

```
data match_merge_yq;
  merge year qtr;
  by id date;
run;
```

Using MERGE with the BY ID DATE statement tells SAS to expect each data set to be sorted (or indexed) by ID/DATE. That requirement is met here. And if there were multiple QTR records with the same ID *and DATE* as a single YEAR record, they would all be matched as desired. That's not the case however - as can be seen in the empty cells below, annual data is not carried forward.

ID	DATE	YR_DATE	AstA	LbtA	SalQ ($mm)	EmpQ (1,000)
XX	30JUN2014	30JUN2014	821	281	132.3	13.5
XX	30SEP2014				128.9	12.6
XX	31DEC2014				138.3	13.5
XX	31MAR2015				112.0	11.3
XX	30JUN2015	30JUN2015	799	303	115.5	11.7
XX	31SEP2015				140.2	14.2
:::	:::	:::			:::	:::
:::	:::	:::			:::	:::
XX	31MAR2017				98.9	9.9

Table 4: Simple Match-Merge Results – Does Not Carry Data Forward

Making a Simple Match-Merge Work

To have a match-merge work, one or both of the time series first needs to be modified, to include a date related variable that would exactly match. For instance, if all fiscal years were calendar years (i.e. ended in December), then you could easily add a CALYEAR variable to each data set (i.e. CALYEAR=YEAR(DATE), and match-merge on ID/CALYEAR, generating a simple one-to-many match. But since not all ID's have fiscal years ending in December, some quarters will be assigned to different calendar years than the annual data. It's better to duplicate annual data, writing out one annual record for each fiscal quarter, with a DATE to match the QTR records:

```
data tempyr (drop=q) / view=tempyr;
  set year;
  yr_date=date;
  do q=1 to 4;
    output;
    date=intnx('month',date,3, 's');
  end;
run;

data qtryr;
  merge qtr (in=inqtr) tempyr;
  by id date;
  if inqtr;
run;
```

This is pretty simple, even though it does require an extra data step (or two extra steps when merging three series). The first step has a loop to output four records per year, using the INTNX function to generate trailing quarterly dates − incrementing three months at a time. However *this technique only works with exact regular series*. If any QTR record fails to fall on a date calculated from the annual dates, the annual variables will be set to missing in the resulting data set, just as in table 4. The real world for time series data will not always be so accommodating[3].

[3] There are other reasons that the real world can complicate this technique. For instance if a firm changes its fiscal year from ending in December to October, then there will be a "short" fiscal year (and fewer quarters to report) for the period preceding the change.

MERGE PLUS CONDITIONAL SET'S: FASTER, SIMPLER

The primary concept in this presentation is this: conditional SET statements can be added to preserve the simplicity of the single step match-merge while carrying forward YEAR data through all of the subsequent QTR records. Like this:

```
data yrqtr;
  merge YEAR (in=iny keep=id date)
        QTR (in=inq keep=id date);
  by id date;

  /*Conditional SET of ALL the year vars*/
  if iny then set YEAR (rename=(date=yr_date));

  /*Conditional Set of all the quarter vars */
  if inq then set QTR;

  if inq then output;

  /* Don't retain data across ID boundaries*/
  if last.id then call missing(of _all_);
run;
```

Here's how it works:

1. The MERGE statement reads only the ID and DATE variables from the two data sets. This means that during every iteration of the data step (i.e. during every merge instance of a YEAR and/or QTR record), all other incoming variables are left unmodified by this statement. That is, all the other incoming variables are, in SAS terminology, "retained".
2. The "in=iny" and "in=inq" data set name parameters tell SAS to make dummy variables INY and INQ indicating whether the current ID/DATE merge has data from the YEAR and/or QTR data set. In the sample data above, for each instance of simultaneous INY=1 and INQ=1, there would also be three instances of INY=0 and INQ=1.
3. The conditional SET: the "if iny then set YEAR …" statement *is only executed when MERGE has indicated incoming data from YEAR*. Only when this SET statement is executed are the annual variables (AstA and LbtA) overwritten – because those variables are not read elsewhere in the data step. The result is that annual variables will remain unmodified over all subsequent QTR records … until the next YEAR record occurs. They are "carried forward".

   This statement also re-reads the DATE variable, renaming it to YR_DATE, a useful indicator of the vintage of the corresponding annual data.
4. Similarly, the "if inq then set QTR" statement only reads in quarterly variables when there is a quarterly record in hand.
5. Note that the MERGE and each SET statement are parallel, synchronized, data streams. When MERGE encounters an annual or quarterly record, the appropriate SET statement reads the very same record – they just read different variables.
6. The conditional output statement assures exactly one output record for each QTR record.

7. Finally, a little housekeeping needs to be done. Once the last record for a given ID has been output, all the variables are set to missing values, to avoid contaminating the next ID.

This data step takes full advantage of the fact that the incoming data sets are sorted. Unlike the PROC SQL it doesn't need to perform a Cartesian comparison of data values, and should always be the most efficient approach. True, the YEAR and QTR datasets are each "read in" twice – once by MERGE and once by SET. But that will not noticeably increase disk activity, since these synchronized data streams will read from a shared buffer established by the operating system.

MORE THAN TWO TIME SERIES? TOO EASY

The simple structure of the above makes expansion to three or more time series a trivial process. For each additional data set, you only need to:

1. Add the dataset name to the MERGE statement, and
2. Add a corresponding ***IF … then*** *SET* statement.

Here's what it looks like for YEAR, QTR, and MONTH, with new or revised code underlined:

```
data YQM ;
  merge YEAR  (in=iny keep=id date)
        QTR   (in=inq keep=id date)
        MONTH (in=inm keep=id date);
  by id date;
  if iny then set YEAR (rename=(date=YR_date));
  if inq then set QTR  (rename=(date=QTR_date));
  if inm then set MONTH (rename=(date=MON_date));
  if inm then output;
  if last.id then call missing(of _all_);
run;
```

Compare the simplicity of this modification to the construction of hierarchical subqueries ("THREE-LEVEL HIERARCHICAL LOCF USING SQL" in the appendix) that seems to be the best SQL-approach.

REMOVING "STALE" DATA USING SENTINEL VARIABLES

A disadvantage of the program as written above is that it does not put an upper limit on how far forward a yearly record is carried. For instance, if a YEAR record is missing for 2015, then (unlike the SQL example) the 2014 annual data would be carried forward for eight quarters, as in table 5:

ID	DATE	YR_DATE	AstA	LbtA	SalQ ($mm)	EmpQ (1,000)
XX	30JUN2014	30JUN2014	821	281	132.3	13.5
XX	30SEP2014	*30JUN2014*	*821*	*281*	128.9	12.6
XX	31DEC2014	*30JUN2014*	*821*	*281*	138.3	13.5
XX	31MAR2015	*30JUN2014*	*821*	*281*	112.0	11.3
XX	30JUN2015	*30JUN2014*	*821*	*281*	115.5	11.7
XX	31SEP2015	*30JUN2014*	*821*	*281*	140.2	14.2

ID	DATE	YR_DATE	AstA	LbtA	SalQ ($mm)	EmpQ (1,000)
:::	:::	:::			:::	:::
:::	:::	:::			:::	:::
XX	31MAR2017	*30JUN2016*	*804*	*322*	98.9	9.9

Table 5: **Result With "Stale" Yearly Data –** Due to Missing YEAR Record for 30JUN2015

The orange cells have annual data that has been carried forward more than three quarters. You might want to consider such instances as stale and set the corresponding annual variables to missing. Logically this requires two actions:

1. DETECT when annual data has become stale (i.e. compare DATE to YR_DATE).
2. SET TO MISSING all the stale variables. That is, use a statement like
   `CALL MISSING(of first_stale_variable -- last_stale_variable);`

The call missing statement tells SAS to set to missing: (1) first_stale_variable, (2) last_stale_variable, and (3) all variables positioned between them - that's what the double dash means. For this data it would be

```
CALL MISSING(of ASTA -- LBTA);
```

But the collection of variables in one or the other dataset commonly changes, which requires monitoring and modifying the call missing statement, a detail far too easily overlooked. Here's how to avoid that problem by positioning sentinel variables just before and after the stale variables:

```
data YQTR_holes (drop=_sentinel:);
  merge YEAR  (in=iny keep=id date)
        QTR   (in=inq keep=id date);
  by id date;
  retain _sentinel1 .;
  if iny the set YEAR (rename=(date=YR_date));
  retain _sentinel2 .;

  if inq then set QTR (rename=(date=QTR_date));

  if YR_date^=. And intck('month',YR_DATE,QTR_date)>11
    then call missing(of _sentinel1--_sentinel2);

  if inq then output;
  if last.id then call missing(of _all_);
run;
```

Why does this work? Because the SAS data step compiler maintains the list of variables (the "program data vector") in the order they are revealed by the code. So variables will be in the following sequence:

ID and DATE	From the MERGE statement
_SENTINEL1	From the first RETAIN
The remaining YEAR vars (YR_DATE, ASTA, LBTA)	From SET YEAR
_SENTINEL2	From the second RETAIN

The remaining QTR vars (QTR_DATE SALQ EMPQ)	From the SET QTR

The values of the _SENTINEL variables are irrelevant. They are merely used as placeholders to enable a compact CALL MISSING when data becomes stale, no matter what the important variables are named. The operational statement is

```
if YR_date^=. and intck('month',YR_DATE,QTR_DATE)>11
  then call missing(of _sentinel1--_sentinel2);
```

The intck function counts months between YR_date and QTR_date. If there are more than eleven months between them, then all the variables from _SENTINEL1 through _SENTINEL2 are set to missing – i.e. all stale annual values are eliminated, resulting in the data below:

ID	DATE	YR_DATE	AstA	LbtA	SalQ ($mm)	EmpQ (1,000)
XX	30JUN2014	30JUN2014	821	281	132.3	13.5
XX	30SEP2014	*30JUN2014*	*821*	*281*	128.9	12.6
XX	31DEC2014	*30JUN2014*	*821*	*281*	138.3	13.5
XX	31MAR2015	*30JUN2014*	*821*	*281*	112.0	11.3
XX	30JUN2015				115.5	11.7
XX	31SEP2015				140.2	14.2
:::	:::	:::			:::	:::
:::	:::	:::			:::	:::
XX	31MAR2017	*30JUN2016*	*804*	*322*	98.9	9.9

Table 6: **"Stale" Yearly Data Removed**

MIXING IRREGULAR SERIES – *EVENT-BASED* LOCF

What if, instead of always carrying lower frequency data forward over multiple higher frequency records, you need to carry data from any source across multiple records from any other source? For instance, consider the three irregular data sets below, each recording a particular type of event: (1) admissions/discharges, (2) services, and (3) tests:

ID	DATE	ACTION	ACTIONMD
101	15JAN2015	A	X23
101	21JAN2015	D	C55
102	15FEB2014	A	X10
102	25FEB2014	D	C99

Table 7: ADMISSIONS/DISCHARGES

ID	DATE	SVCID	CHGUNITS

101	18JAN2015	323	3.2
101	19JAN2015	488	1.2
102	15FEB2014	101	3.0
102	16FEB2014	229	1.7

Table 8: SERVICES

ID	DATE	TSTLAB	TSTTYPE
101	20JAN2015	788	VIS
102	16FEB2014	823	HRT

Table 9: TESTS

Unlike SQL, the code for carrying forward event-based data is just as simple as the hierarchical LOCF programs above:

```
data event_carried_forward;
  merge admdis (in=inA keep=id date)
        srvcs  (in=inS keep=id date)
        tests  (in=inT keep=id date);
  by id date;
  if inA then set admdis (rename=(date=date_A));
  if inS then set srvcs  (rename=(date=date_S));
  if inT then set tests  (rename=(date=date_T));

  output;   /* Output a record for every event. No IF test needed*/
  if last.id then call missing (of _all_);
run;
```

which produces (yellow italics bold are "carried forward" data):

ID	DATE	DATE_A	DATE_S	DATE_T	A/D	AID	SID	CHG	LAB	LTYPE
101	15jan2015	15jan2015			A	X23				
101	18jan2015	***15jan2015***	18jan2015		***A***	***X23***	323	3.2		
101	19jan2015	***15jan2015***	19jan2015		***A***	***X23***	488	1.2		
101	20jan2015	***15jan2015***	***19jan2015***	20jan2015	***A***	***X23***	***488***	***1.2***	788	VIS
101	21jan2015	21jan2015	***19jan2015***	***20jan2015***	D	C55	***488***	***1.2***	***788***	***VIS***
102	15feb2014	15feb2014	15feb201		A	X10	101	3.0		
102	16feb2014	***15feb2014***	16feb2014	16feb2014	***A***	***X10***	229	1.7	823	HRT
102	25feb2014	25feb2014	***16feb2014***	16feb2014	D	C99	***229***	***1.7***	***823***	***HRT***

Table 10: Event-Based Data Carried Forward

The SQL equivalent of this ("EVENT-BASED LOCF FOR THREE SERIES USING SQL" in the appendix) is requires an initial union of all dates, followed by specification of two views, before generating a table equivalent to table 10. It's not gruesome, but it is tedious.

CARRY THE FUTURE BACK: HIERARCHICAL NOCB

The same conditional SET technique can also be used to carry future observations back, without wasting resources in applying descending sorts (for every data set!). It just requires modifying the low frequency[4] data, generating a data set that looks like table 11:

ID	YR_DATE	MATCHDATE	AstA ($mm)	LbtA ($mm)
XX	30JUN2014	.	821	281
XX	30JUN2015	01JUL2014	799	303
XX	30JUN2016	01JUL2015	804	322
YY	31DEC2014		1,401	904
YY	31DEC2015	01JAN2015	1,427	950
YY	31DEC2016	01JAN2016	2,550	962

Table 11: NOCB-Ready Fiscal **YEAR Data (2 ID's, 3 Years)**

Note the original DATE value is stored in YR_DATE, and the new variable MATCHDATE takes the value of one day after the prior YR_DATE. Once this is done, then *carrying back data* from a given YR_DATE to all quarters since the preceding YR_DATE *is equivalent to carrying forward data* from the preceding MATCHDATE. So once the MATCHDATE variable is created, the logic of the LOCF programs above can be used.

Here's how, in two steps:

```
data yfuture/view=yfuture;
  set year (rename=(date=yr_date));
  by id;
  matchdate=lag(yr_date+1);  /*1 day after prior annual record*/
  if first.id then matchdate=.;
  /*matchdate=max(matchdate,intnx('month',yr_date+1,-18,'same'));*/
run;

data year_carried_back (drop=_sentinel: matchdate);
  merge qtr     (in=inq keep=date id rename=(date=matchdate))
        yfuture (in-iny keep=date id);
  by id matchdate;
  if inq then set qtr;

  retain _sentinel1 .;
  if iny then set yfuture;
  retain _sentinel2 .;

  if yr_date<date then call missing(of _sentinel1 -- _sentinel2);
  if inq then output;
  if last.id then call missing(of _all_);
run;
```

The first step creates the yfuture data set *view* (I could make it a data set *file*, but why waste disk activity?), with the new backdated MATCHDATE variable described above. When a new ID is encountered (i.e. first.id=1) MATCHDATE is set to missing to avoid

[4] Or in the case of more than two series, modify all but the highest frequency. I.e. modify both YEAR and QTR when mixing them with MONTH.

contamination with dates from the prior ID. The resulting YEARFUTURE data set is shown in Table 11 above.

If you want to prevent data from being carried back too far, just de-comment the statement following "if first.id …". This prevents annual data from being carried back more the 18 months. It's the NOCB analog of removing stale data for LOCF programs.

Once data set YFUTURE is create, the second step uses coding virtually identical to the LOCF programs, but now annual data is carried back to preceding quarters, producing data in table 12:

ID	DATE	YR_DATE	AstA	LbtA	SalQ	EmpQ
XX	30JUN2014	30JUN2014	821	281	132.3	13.5
XX	30SEP2014	*30JUN2015*	*799*	*303*	128.9	12.6
XX	31DEC2014	*30JUN2015*	*799*	*303*	138.3	13.5
XX	31MAR2015	*30JUN2015*	*799*	*303*	112.0	11.3
XX	30JUN2015	30JUN2015	799	303	115.5	11.7
XX	30SEP2015	*30JUN2016*	*804*	*322*	140.2	14.2
XX	31DEC2015	*30JUN2016*	*804*	*322*	107.7	10.6
XX	31MAR2016	*30JUN2016*	*804*	*322*	128.6	13.0
XX	30JUN2016	30JUN2016	804	322	127.8	13.0
XX	30SEP2016				130.0	12.9
XX	31DEC2016				120.6	11.8
XX	31MAR2017				98.9	9.9
YY	31DEC2014	31DEC2014	1,401	904	155.3	15.5
:::	:::	:::	:::	:::	:::	:::

Table 12: Annual Data Carried Back to Preceding Quarterly Records (ID XX Only)

Note there is no annual data for id XX after 30JUN2016, which results in the blank green cells. But for all the prior years, annual data is carried back to preceding quarterly records, in the blue cells.

IRREGULAR SERIES NEXT OBSERVATION CARRIED BACK

Creating a MATCHDATE for each dataset of irregular series provides the same capability of carrying back future date from event-based series. The program below does exactly that for the ADMDIS, SRVCS, and TEST datasets from earlier in this paper:

```
data va  /view=va;
  set admdis;
  by id;
  matchdate=lag(date+1);
  if first.id then matchdate=.;
  /* De-comment next line if limiting carry-back to 3 months max */
  /* matchdate=max(matchdate,intnx('month',date+1,-3,'same'))*/;
```

```
    output;

    if last.id then do; /*Make a dummy final "future" record*/
      array tmp {2} _temporary_;    tmp{1}=id;  tmp{2}=date;
      call missing (of _all_);
      id=tmp{1};
      matchdate=tmp{2}+1;
      output;
    end;
  run;

  data vs / view=vs;
    /* similar code using dataset SRVCS */
  run;
  data vt / view=vt;
    /* similar code using dataset TEST  */
  run;

  data event_nocb (drop=matchdate label='Event-Based Next Obs Carried Back');
    merge va (in=ina keep=id matchdate)
          vs (in=ins keep=id matchdate)
          vt (in=int keep=id matchdate);
    by id matchdate;
    format date date9.;
    if ina then set va (rename=(date=date_A));
    if ins then set vs (rename=(date=date_S));
    if int then set vt (rename=(date=date_T));
    date=min(of date_:);
    if date^=.;   /* Don't output when all components are dummy records */
  run;
```

which produces these results:

ID	DATE	DATE_A	DATE_S	DATE_T	A/D	AID	SID	CHG	LAB	LTYPE
101	15jan2015	15jan2015	*18jan2015*	*20jan2015*	A	X23	*323*	*3.2*	*788*	*VIS*
101	18jan2015	*21jan2015*	18jan2015	*20jan2015*	*D*	*C55*	323	3.2	*788*	*VIS*
101	19jan2015	*21jan2015*	19jan2015	*20jan2015*	*D*	*C55*	488	1.2	*788*	*VIS*
101	20jan2015	*21jan2015*		20jan2015	*D*	*C55*			788	VIS
101	21jan2015	21jan2015			D	C55				
102	15feb2014	15feb2014	15feb2014	*16feb2014*	A	X10	101	3.0	*823*	*HRT*
102	16feb2014	*25feb2014*	16feb2014	16feb2014	*D*	*C99*	229	1.7	823	HRT
102	25feb2014	25feb2014			D	C99				

Table 13: Event-Based Data Carried Back
(Blue Cells are Carried Back)

CONCLUSION

SAS has a number of tools for dealing with single data sets for time series (e.g. PROC TIMESERIES, PROC EXPAND), but does not have a procedure for mixing time series from multiple sources with differing frequencies. What SAS does have, though, is the DATA step, with a SET statement (or MERGE) that can easily be applied to the task of carrying time series data forward. That makes it an excellent tool for generating a "snapshot" of mixed series data at any time point. In most cases, it is a far better tool than PROC SQL for last-observation-carried-forward processes. The specific points shown in this paper are:

1. The SET statement, when issued conditionally, makes it easy to carry forward data from one observation to the next. This provides a straightforward way to mix time series of different frequencies.
2. This technique is easily scaled up to mixing several series.
3. Stale data can easily be removed by using "sentinel" variables on each side of a set of variables at risk of becoming "out-of-date". Using those sentinel variables in a CALL MISSING(of …) statement allow you to avoid the need to specify lists of stale variables by name.
4. The advantages of the conditional SET approach is especially apparent when applied to irregular series. Unlike the SQL solution, the code for mixing irregular series is almost identical to that for mixing hierarchical series, involving nothing more than the removal of an IF condition in the OUTPUT statement.
5. With minor preparation of datasets, the same program structure can be used to carry back future data without descending sorts.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Mark Keintz
mkeintz@wharton.upenn.edu

APPENDIX

THREE-LEVEL HIERARCHICAL LOCF USING SQL:

```
/* Adding levels to the hierarchy requires subquery expressions */
proc sql ;
  create table sql_yqm as select * from
  month as m left join
    (select * from qtr (rename=(date=qtr_date)) as q
        left join year (rename=(date=yr_date))  as y
    on y.id=q.id and yr_date<=qtr_date
    group by q.id,qtr_date having yr_date=max(yr_date)
    ) as yq
  on m.id=yq.id  and QTR_DATE<=date
  group by m.id,date having QTR_DATE=max(QTR_DATE);
quit;
```

EVENT-BASED LOCF FOR THREE SERIES USING SQL:

```
proc sql ;
  create table datelist as
    select id,date from admdis   union
    select id,date from srvcs    union
    select id,date from tests;

  create view sql_evt_a as select * from
    datelist as d  left join  admdis (rename=(date=date_a)) as a
    on d.id=a.id and date_a<=date
    group by d.id,date having date_a=max(date_a) ;

  create view sql_evt_s as select * from
    datelist as d  left join  srvcs (rename=(date=date_s)) as s
    on d.id=s.id and date_s<=date
    group by d.id,date having date_s=max(date_s) ;

  create table sql_locf_evt as select * from
    (select * from sql_evt_a as a join sql_evt_s as s
     on a.id=s.id and a.date=s.date) as as
    join
    (select * from  datelist as d
     left join  tests (rename=(date=date_t)) as _t
     on d.id=_t.id and date_t<=date
     group by d.id,date having date_t=max(date_t)) as t
    on as.id=t.id and as.date=t.date;
quit;
```