PharmaSUG 2019 - Paper BP-302

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

Mark Keintz, Wharton Research Data Services

## ABSTRACT

Many programming tasks require merging time series of varying frequency. For instance you might have three datasets (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 may 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 (i.e. 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'll also show how to use "sentinel variables" as a means of controlling the maximum time-span data is carried forward, i.e. how to remove historical data that has become "stale." Finally, we will demonstrate how to modify these techniques to carry future observations backward, without re-sorting data.

## INTRODUCTION

The principal objective of this presentation is to show that a single well-designed DATA step can be much more efficient at last-observation-carried forward tasks than the usual proc sql or multi data step approaches. It takes advantage of a single underlying property of the SET statement – namely that variables read by SET are not reset to missing in every iteration of the DATA step. Instead those variables are "retained", i.e. they keep the most recent values until that SET statement is encountered again.[1] The examples in this presentation will show how to conditionally read from each input data set to trivially carry forward values from low frequency series (e.g. YEAR) while reading higher frequency series (e.g. QTR). 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 |
| XX | 30JUN2015 | 799 | 303 |
| XX | 30JUN2016 | 804 | 322 |
| YY | 31DEC2014 | 1,401 | 904 |
| YY | 31DEC2015 | 1,427 | 950 |

[1] This is also a property of the MERGE statement, but discussion here will focus on using the SET statement.

| | | | |
|---|---|---|---|
| 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 a new variable YR_DATE, the DATE value taken from the YEAR data set

| 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 SOLUTION OR MULTIPLE DATA STEPS

Commonly this task is handled either through SQL code or multiple data steps. Each can solve the problem, but each also has some deficiencies.

### THE PROC SQL APPROACH

For instance this code is a likely PROC SQL solution:

```
proc sql noprint;
  create table yrqtr as
  select *
    from qtr
```

```
  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 annual date can neither follow the quarterly date nor precede 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 each additional subquery can create a major performance hit.
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.

## THE MULTIPLE DATA STEP APPROACH

At first you might imagine that multiple data steps are not needed, and that a standard match-merge (MERGE plus BY statements) might 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. And if there were multiple QTR records with the same ID/DATE as a single YEAR record, they would all be matched as desired. That's not the case however, and 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**

To have a match-merge work, one or both of the time series would first need to be modified, to include a date related variable that would 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, and match-merge on ID/CALYEAR. But in this case, not all ID's have the same fiscal year. It would be better to duplicate the annual data, writing out one 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 regular series. If any QTR record fails to fall on a date calculated from the annual date value, 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.

## MERGE PLUS CONDITIONAL SET STATEMENTS: FASTER AND SIMPLER

As stated in the introduction, 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 carry data across ID boundaries*/

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

Here's how it works:

1. The MERGE statement only reads in the ID and DATE variables from the two data sets. This means that during every iteration of the data step (i.e. during every read of a YEAR and/or QTR record), all other incoming variables are left unmodified by this statement.

2. The “in=iny” and “in=inq” data set name parameters tell SAS to make dummy variables INY and INQ to indicate whether the current ID/DATE merge has data from the YEAR and/or QTR data set. For the sample data above, for each instance of simultaneous INY=1 and INQ=1, there would 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) are overwritten – because those variables are not read in 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.

   This statement also rereads the DATE variable, renaming it to YR_DATE.
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 operating system.

## MORE THAN TWO TIME SERIES – EASILY DONE

The simple structure of the above makes expansion to three or more time series a simple 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:

```
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;
```

## REMOVING “STALE” DATA USING SENTINEL VARIABLES

In the program above, if a YEAR record is missing for 2015, then the 2014 annual data would be carried forward for eight quarters (an outcome not produced in the SQL example), 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 |
| ::: | ::: | ::: | | | ::: | ::: |
| ::: | ::: | ::: | | | ::: | ::: |
| 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 four 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 YR_DATE to 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 first_stale_variable, last_stale_variable, and all variables positioned between them - that's what the double dash means. In this example it would be

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

But in the presence of changing variable names, this requires monitoring and modifying the call missing statement, a bit too easy to overlook. The program below shows 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 then set YEAR (rename=(date=YR_date));
  retain _sentinel2 .;

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

  if YR_date^=. and intck('qtr',YR_date,QTR_date)>3
    then call missing(of _sentinel1--_sentinel2);

  if inq then output;
  if last.id 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, the variables would 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 _SENTINEL variables are merely used as placeholders to enable a CALL MISSING when data becomes stale. The operational statement is

```
if YR_date^=. and intck('qtr',YR_date,QTR_date)>3
  then call missing(of _sentinel1--_sentinel2);
```

The intck function counts quarters between YR_date and QTR_date. If there are more than three quarters between them, then all the variables from _SENTINEL1 through _SENTINEL2 are set to missing – i.e. all stale annual values (except YR_DATE) 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 | ***30JUN2014*** | | | 115.5 | 11.7 |
| XX | 31SEP2015 | ***30JUN2014*** | | | 140.2 | 14.2 |
| ::: | ::: | ::: | | | ::: | ::: |
| ::: | ::: | ::: | | | ::: | ::: |
| XX | 31MAR2017 | ***30JUN2016*** | ***804*** | ***322*** | 98.9 | 9.9 |

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

## MIXING IRREGULAR DATA SERIES – EVENT-BASED DATA CARRIED FORWARD

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**

The code for carrying forward event-based data is as simple as the 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 (including the extra dates):

| 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**

## CARRY THE FUTURE BACK: MATCH QTR WITH FOLLOWING ANNUAL DATA

The same principle of using conditional SET statements can also be applied to the task of carrying future data back, without wasting resources on the common strategy of applying descending sorts (for every data set!). It does require modifying the low frequency (i.e. YEAR) data set[2], creating a data set that looks like table 11. Note that the original DATE value is stored in YR_DATE, and the new value for DATE is assigned to be one day after the prior YR_DATE. After this modification, the logic of the examples above can be used:

| ID | YR_DATE | DATE | 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: Modified Fiscal YEAR Data (2 ID's, 3 Years)**

Here's how, in two steps:

```
data yearfuture/view=yearfuture;
  set year (rename=(date=yr_date));
  by id;
  date=sum(lag(yr_date),1);  /*Backdate DATE to 1 day after prior record*/
  format date date9.;
  if first.id then date=.;
run;

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

  retain _sentinel1 .;
  if iny then set yearfuture;
  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 yearfuture data set view (I could make it a data set FILE, but why waste disk activity?). It has the new backdated DATE variable, by adding 1 day to the lagged value of yr_date (i.e. one day after the prior value of yr_date). When a new ID is encountered (i.e. first.id=1) avoid using dates from the preceding id by resetting date to a missing value. Or you could assign an arbitrary value to date

[2] 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.

every time you start a new ID. For instance, you could backdate by 18 months minus one day:

```
if first.id then date=intnx('month',yr_date,-18,'s')+1;
```

You can avoid backdating by too much when there is a hole in the annual data, with a statement like `if intck('month',date,yr_date)>18 then date=intnx('month',yr_date,-18,'s')+1;` which forces the backdating to be no greater than 18 months. The resulting YEARFUTURE data set is shown in Table 11 above.

At that point the second step uses coding virtually identical to the 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.

## CONCLUSION

When faced with mixing data of differing time series, SAS users often resort to inefficient SQL techniques or unnecessary extra data steps to combine those series. This can almost always be improved upon by using a data step that takes advantage of the following properties:

1. The SET statement, when issued conditionally (i.e. in an IF statement), 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, whether regular or irregular.
2. Stale data can easily be removed by using “sentinel” variables on each side of a sequence of variables to be set to missing when they are deemed out of date. Using those sentinel variables in a CALL MISSING(of …) statement allow you to avoid he need to specify lists of stale variables by name.

3. With minor preparation of datasets, the same program structure can be used to carry back future data.

## CONTACT INFORMATION

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

Mark Keintz
Wharton Research Data Services
mkeintz@wharton.upenn.edu