Moving dates around

A simplistic way to work with dates with monthly data in SAS is to convert all dates to a year and month, and then use those. For instance

```
data msf;
   set crsp.msf(keep=permno date ret);
   year=year(date);
   month=month(date);
run;
proc sort data=msf;
   by permno year month;
run;

proc means data=msf;
   by permno year;
   var ret;
run;
```

This is appealing when, say, you have to calculate within-year averages, but it can quickly get tedious if you have to do other simple tasks. For instance, suppose you want to create a series of *daily* dates from January 1, 1961 to December 31, 1962.

```
data dates;
    do year=1961 to 1962;
       do month=1 to 12;
          lengthofmonth=31;
          if month=4 or month=6 or month=9 or month=11 then lengthofmonth=30;
          if month=2 then lengthofmonth=28;
          do day=1 to lengthofmonth;
             output;
          end;
       end;
     end;
run;
```

This is terribly tedious - not to mention fraught with the danger that you'll make errors in leap years. The alternative is to use SAS' intnx function, which increments dates by specified intervals.

```
data dates;
    date='01Jan1961'd;
    do while(date<='31Dec1962'd);
        output;
        date=intnx('day', date, 1);
     end;
run;
```

The intnx function increments date (its second argument), by 1 unit (the third argument) of 'day' (its first argument).

The more astute among you will observe that I get the same effect by saying

```
data dates;
    date='01Jan1961'd;
```

```
        do while(date<='31Dec1962'd);
            output;
            date=date+1;
         end;
    run;
```

because dates are treated as numbers in SAS. This is well and good if I want to increment by single days. Suppose instead I want to increment by weekdays. That is, I want the list of all weekdays in 1961 and 1962. An easy way to do this is:

```
    data dates;
        date='01Jan1961'd;
        do while(date<='31Dec1962'd);
            output;
            date=intnx('weekday', date, 1);
         end;
    run;
```

which is a useful construct in working with the DSF file, and horrendously complicated to do any other way.

The intnx function, as already said, increments dates, but it does so in a nonintuitive way. Suppose you are incrementing by year.

```
    date=intnx('year', date, 1);
```

Intnx moves the date to the first day of the subsequent year, not forward by one twelve-month period. If you say 'month' instead of 'year', intnx moves the date forward to the first day of the next month however many times you specify. Saying

```
    date=intnx('month', date, 0);
```

moves the date to the beginning of the month - the first of the month- in which date falls. Similarly for 'qtr'.

I commonly use the intnx function to do the following:

- Move the date to the end of the month/quarter/year

  ```
  date=intnx('month', date, 1)-1;
  date=intnx('qtr', date, 1)-1;
  date=intnx('year', date, 1)-1;
  ```

- Move the date to the beginning of the month/quarter/year

  ```
  date=intnx('month', date, 0);
  date=intnx('qtr', date, 0);
  date=intnx('year', date, 0);
  ```

- Move the date to the end of the previous month

  ```
  date=intnx('month', date, 0)-1;
  ```

- Move the date (which is an end-of-month) to the end of the next month

  ```
  date=intnx('month', date+1, 1)-1;
  ```

  Observe here that I have to say date+1. Since the date is currently of the form 31Jan1999, saying

  ```
  date=intnx('month', date, 1)-1;
  ```

will leave us with 01Feb1999-1, or 31Jan1999.

- Move the date to the end of month 12 months ago

  ```
  date=intnx('month', date, -11)-1;
  ```

- Move the date (which is an end-of-month) to the end of the 12 months ahead

  ```
  date=intnx('month', date+1, 12)-1;
  ```

- Given a beginning date and an ending date, create a dataset with all the month-ends in between:

  ```
  data beg_end;
     beginning_date='31Jan1996'd;
     ending_date='31Dec2007'd;
  run;
  data all_dates;
     set beg_end;
     date= beginning_date;
     date=intnx('month', date, 1)-1;
     do while(date<=ending_date);
        output;
        date=intnx('month', date+1, 1)-1;
     end;
  run;
  ```

  The first

  ```
     date=intnx('month', date, 1)-1;
  ```

  statement is to ensure that the first date is an end-of-month.

- Produce a list of all Wednesdays between two dates:

  ```
  data beg_end;
     beginning_date='31Jan1996'd;
     ending_date='31Dec2007'd;
  run;

  data all_dates;
      format date date9.;
      set beg_end;
      date=beginning_date;
      if weekday(date)>4 then date=intnx('week', date, 1)+3;
      else date=intnx('week', date, 0)+3;
      do while(date<=ending_date);
          output;
          date=intnx('week', date, 1)+3;
          end;
  run;
  ```

  The weekday() function returns the day of the week, 1 being Sunday. I make the first date a Wednesday (weekday=4) with

  ```
      if weekday(date)>4 then date=intnx('week', date, 1)+3;
      else date=intnx('week', date, 0)+3;
  ```

  The logic is simple. If the current date is ``more'' than a Wednesday (ie, weekday(date)>4), then the first admissible date is Wednesday of next week. Sunday of next week is

  ```
   date=intnx('week', date, 1);
  ```

Add 3 to that to get to Wednesday. If the current date is ``less than or equal to'' Wednesday, then go to the beginning of this week

```
date=intnx('week', date, 0);
```

and add 3 to that.

The rest of the scheme is obvious. This is useful in constructing weekly returns (Thursday-to-Wednesday to avoid any weekend effect).

Andre de Souza 2012-11-19