ABSTRACT

There are times in most SAS® programmers’ lives when they wish they could move freely around their data, unobstructed by the linear nature of datasets. They feel forced to lag() behind other programmers, never looking forward to their next observation.

To these poor unfortunates I present POINT. When used correctly, it provides the flexibility to move through datasets: looking ahead, starting over, jumping into the middle of it.

INTRODUCTION

When you think of data in the realm of SAS, you usually think in terms of sequential processing: one observation after the other. If the need to keep information from previous observations arises, the retain statement or the lag function can be used. These options are reasonably straightforward and familiar to most programmers.

When the situation calls for looking ahead for information in later observations, you usually have to resort to some unique solution. This may include such things as reverse sorts and re-merging data, or setting some sort of special flag and rereading the dataset.

By taking advantage of the POINT argument of the SET statement in the DATA step, you can move back and forth through datasets at will. This allows you to tackle problems with novel approaches that you may have thought were not available in SAS.

HOW IT WORKS

The POINT argument is pretty much what its name implies: a pointer. The following example shows a simple use of the POINT argument: it stores the fifth observation of dataset Y in dataset X.

```sas
DATA X;
   RETAIN _PTR 5;
   SET Y POINT = _PTR;
   IF _PTR > 5 THEN STOP;
   _PTR = _PTR + 1;
RUN;
```

SAS opens the dataset Y at the observation number five (the value contained in the variable _PTR). This allows you to “point” at an observation. The power to move around the dataset is in your hands. By incrementing or decrementing the pointer, you can move through the dataset.

By setting up the pointer as a retained variable, and carefully manipulating the variable, you can move throughout the dataset. Specifying _PTR = _PTR + 5 will move you five observations ahead. Specifying _PTR = _PTR – 5 will move you back five observations.

Since you have now taken control of a function SAS normally does for you, you need to be aware of the overhead required for using this option. The SAS system no longer automatically moves to the next observation, it is up to you to tell the system where to go. By using the RETAIN with the pointer and incrementing it as needed, you can control the processing through the dataset.

The IF condition in the code above is an example of the overhead of the POINT argument. The dataset will loop back, SET the dataset Y, POINT at observation number six, and continue to the end of the dataset. When it gets to the end of the dataset, it will still try to read data. Since you are in control of the data manipulation, you have to tell the datastep when to end, or it will continue to try to read the dataset Y forever. By specifying the condition to stop processing, you control the end of the datastep.

HOW TO USE IT

The POINT argument can be used in most places where a merge would normally be used. By using multiple SETs, you can replace the merge. However, if the merge is a straightforward one-to-one or many-to-one merge then a MERGE statement should be used.

The example at the end of the paper shows how to use the POINT argument to collect dates that would not necessarily be easy to collect using standard methods. By presenting the data, as well as highlighting the processing, I hope to show how POINT can be used to solve this problem.

Let us start by assuming you have an adverse events dataset (Dataset A) that has dates of adverse events, and a dosing dataset (Dataset B) that has the date a study drug was taken and its dose. You need to find the dose that the patient was on when an adverse event occurred.

If the adverse event occurred before the dosing started, then the doseflag value should be set to missing. If the patient was on study drug, which can change over the course of the study, we want to capture the dose closest to, but not after the adverse event took place. For our example, the Dataset C shows the desired results.

Looking at the code at the end of the paper, you can follow the comments below with the comment numbers at the right of the code. The eight comments highlight the information needed to use the POINT argument. The code is presented only to illustrate the use of the POINT argument.

The first step is to rename the variables contained in the dataset to be SET with the POINT argument (See comment 1). This is not necessary if they are named different than the variables in the other dataset.

Comment 2 shows the initialization of the POINT variable. It is done as a macro variable to avoid any conflict with other dataset variables, and to allow a sort of “retain” by using the SYMPUT and SYMGET functions. Comment 3 shows the SYMGET used to bring the variable into the datastep.

Comment 4 shows the actual SET and POINT. At this point in time the dataset all has two datasets that are being merged under its control (ADVERSE and DRUG), and a third dataset that you control (DRUGMERG). Retaining the POINT variable is essential for moving throughout the third dataset.

The lines around comment five show the logic of our “manual merge”. If the patient and investigator are the same as those in the adverse event dataset, then this logic retains the dose until the dose date is after the adverse event date.

Comment 6 shows the manual incrementing of the pointer that you are responsible for. The _point variable is the only mechanism for moving through the DRUGMERG dataset. This is how you control the datastep flow.
Comment 7 shows the tests needed to stop the loop. Remember that you have to let the datastep know when to stop running. In this case there is a check for when the patient changes, as well as an end of file check.

Be aware that you not only control the looping in this example, but that you also define the looping. There is a do loop surrounding the SET statement that actually controls that SET statement. This is how you can read any number of observations on the DRUGMERG dataset without moving to the next observation on the DRUG/ADVERSE dataset.

The final comment (8) shows the storing of the pointer information for the next investigator/patient combination. The SYMPUT with the decremented pointer allows for multiple adverse events on the same day.

**HOW IT CAN BE MISUSED**

Although the POINT argument gives you a great deal of freedom to manipulate data, it requires more of you than a simple SET. There are four things that you have to be aware of when you use this argument. The first three deal with the responsibility you have to control the process, and the final deals with the functionality you lose when you take that control from the data step.

The first responsibility you have is to initialize the variable for the POINT argument. You are responsible for telling the SET where to start reading. If the variable is incorrectly set, the results of the SET can be unpredictable. Also be aware that the variable you choose should be unique for the datasets used in the data step.

You have to specify the condition that will end the SET. The POINT argument does not work with the standard END= argument. The easiest thing to forget when using the POINT argument is that you have to tell the datastep when to stop reading the dataset. If you do not specify a terminating condition, then the SET will continue reading records until the job is stopped.

Besides having to control where to start and when to stop, you also have to control the movement through the dataset. Once the variable used to point is initialized, you have to modify it to move to the observation that you want to be at. If the pointer is not changed, the save observation will be read again and again until the job is stopped.

The final point about POINT is that the SAS automatic variables FIRST. and LAST. cannot be used. Since you are controlling the access to the data, you cannot use the BY statement for that SET. Any sorting must be done before the data step.

**CONCLUSION**

For special cases, and when care is used, the POINT argument is a very powerful tool. When you need to move through the DATA step in ways that are not standard in SAS, such as looking ahead for observations, the POINT option is an obvious choice.

When you need to look through parts of a dataset without reading all of the dataset, this command become very useful. In cases where the observations are thousands or millions of observations in the dataset, being able to point to their location can save a great deal of time.

By thinking of datasets as random access tables, you can use the POINT argument to move through the data in ways that can simplify your programming tasks.

**REFERENCES**

SAS Institute, Inc. (1990), SAS® Language Reference, Version 6,
### Example Datasets

<table>
<thead>
<tr>
<th>REC</th>
<th>DATE</th>
<th>DOSE</th>
<th>DATE</th>
<th>AMOUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>01NOV1997</td>
<td></td>
<td>06NOV1997</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>11NOV1997</td>
<td></td>
<td>10NOV1997</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>23NOV1997</td>
<td></td>
<td>13NOV1997</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>15NOV1997</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>17NOV1997</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>20NOV1997</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>21NOV1997</td>
<td>300</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>22NOV1997</td>
<td>300</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>24NOV1997</td>
<td>100</td>
</tr>
</tbody>
</table>

Dataset A

Dataset B

Dataset C

### Code example for POINT= argument

```sas
proc sort data = data.drug
   out = drug (keep= invno patient visitid firstdt stopdt dose dosedt);
   by invno patient dosedt;
run;

*** Get the number of observations for drgr dataset. ;
%nobs (drug) ;

*** Rename all variables for a set. ;
data drugmerg (rename = (invno = invnod patient = patientd
dose = dosed dosedt = datedose)) ;
   set drug (keep = visitid invno patient dose dosedt) ;
   by invno patient dosedt ;
run;

proc sort data = data.adverse
   out = adverse (keep = invno patient aetext startdt stopdt);
   by invno patient startdt;
run;

*** Pointer for first record. ;
%let startpt = 1;

data all;
   merge adverse (in = in1)
       drug;
   by invno patient;
   if (in1) ;
   if   (startdt < firstdd) then do ;
      aedose=.;
   end ;
   else if (startdt > (lastdd + 2)) then do ;
      aedose = olddose;
   end ;
   else if (startdt >= firstdt) then do ;
      retain _done ;

   *** Start the pointer near the end of the last search. ;
   _point    = input(symget('startpt'), 6.) ;
   _done     = 0 ;
   do until (_done) ;
```

1

2

3
*** Open up the dataset to search for doses. ;
   set drugmerg point = _point ;  
4

*** Retain the pointer to the observation, and the dose info. ;
   retain _point lastdose ;

*** Only check if patient and investigator are equal. ;
   if (patient = patientd and invno = invnod ) then do ;

*** If the aedate is greater than or equal to the dose date, ;
*** keep the dose info. ;
   if (startdt >= datedose) then lastdose = dose ; 5

*** If the aedate is less than the dose date, then leave, 
*** since we have met criteria with the prior record. ;
   else leave ;
   end ;

*** Move pointer to the next record in the dose dataset. ;
   _point = _point + 1 ;
6

*** Check to see if we need to continue looking. Set flag if not. ;
   if (patient < patientd and invno = invnod) then _done = 1 ;
   if (_point = &nobs) then _done = 1 ;
   end ;
7

*** Drop the pointer back a space incase we have duplicate keys,
*** and store for the next ae. ;
   if (_point >= &nobs) then _point = 2 ;
   call symput('startpt', put((_point - 1), 6.)) ;
8

*** Save the data into the correct variable. ;
   aedose = lastdose ;
   end ;

*** Message if conditions not met. ;
   else do ;
   aedose = -99 ;
   end ;
run ;