Abstract

This paper postulates that the learning process is a matter of strategic importance and discusses ways to learn new SAS® system software more easily. Drawing on examples taken from Screen Control Language, the paper first attempts to show that there is a similarity between new and old programming constructs in SAS system software. Instead of being overwhelmed by new technology, the paper suggests that people may understand it more easily if they can see it in terms of a continuum of software evolution. Improving the learning process does not end there. The paper points out the importance of language in understanding more advanced capabilities of new technology. The use of clear terminology that corresponds closely to functionality is essential. It is hoped that strengthening the learning process in these and other ways will help facilitate the use of SAS system software at different levels of application development.

Introduction

Since it was first released in the 1970's, SAS system software has been evolving at a breathtaking rate. This prolific development poses a growing and critically important educational challenge. To narrow the gap between the introduction of new SAS system software and its deployment in production systems, users must learn to work with the new software quickly. But people need time and energy to get up to speed with the new software, and the real world imposes constraints on both.

Historically, the SAS system user community has awaited new releases of software with eager anticipation knowing that these releases contain attractive new features and powerful programming constructs. As time has gone on, however, the volume and richness of new material is becoming more than many people can assimilate easily. There is almost too much 'good news'. Production is ahead of consumption.

Strengthening the learning process can help alter that imbalance and facilitate the use of the SAS system. Focusing on this subject means drawing attention to what is clearly a matter of strategic significance - for programmers, consultants, teachers, technical support, and sales and marketing experts.

Analytical Framework

What can make it easier for people to learn things about the SAS system they find difficult? There are all kinds of courses and manuals available. Objectively, one could mention these and many other technical information sources, if that is the way one chose to answer the question of improving understanding. For example, SAS system users have access to on-line help including the INTERNET, Institute-sponsored and other training courses, even books by users. They can get superb technical support. There is no
This paper is not about those objective resources. Instead, it contends that the heart of this question is subjective, namely how people initially perceive new SAS System technology and learn it. The question "how do people learn the SAS system?" isn't answered by pointing out the available resources. It requires analyzing how people understand. That is a large subject and this paper covers only a small portion of it.

Please note that this paper also is not a basic primer on the topics mentioned in the title, even though some of the examples may help serve that purpose. It is, instead, a conversation about pedagogy and assumes some familiarity with the subject matter. The ideas presented here are intended to initiate a dialogue on education. It is hoped that other people may add their own insights into the discussion.

Two ideas will be explored here. The first is that some people may be better able to grasp the new if they can relate it to what they know. "New" SAS System technology may have antecedents in "old" SAS System technology that people understand and work with already. They have worked hard to learn the "old" technology and have acquired insights that may be transferable to the "new" technology. In other words, past experience, not just new experience, can be the best teacher.

The second idea is that terminology sometimes gets in the way of understanding. Technical expressions may only approximate meaning. When they do, one must search for underlying meanings and employ language that assists people in developing an insightful relationship into the subject matter.

The author took an informal poll to find some illustrative material for these ideas, using Screen Control Language as an example of new technology and as a general starting point. He was advised to talk about lists, subclasses, and extended tables. For some people, he was told, focusing on these topics is like filling out a 1040 form - something to be avoided, or something that is better left to experts and so-called "gurus".

**Drawing on Past Experience**

**LISTS**

Screen Control Language lists had a modest beginning in early releases of SAS/AF® software. Early list statements such as LISTC and SHOWLIST produced simple-to-use pop-up selection windows. Lists were simply a set of choices displayed on a terminal. In later releases, SCL lists evolved into a new form beyond screen displays. They became available also as programmable data structures having considerable potential. Suddenly there was a lot to learn.

The SCL lists used for programming are often introduced in terms of their similarity to arrays. Why this analogy? Because, it is pointed out, items contained both in lists and in arrays can be referenced by means of their index position. For a few people, this analogy may be perfectly intuitive, but it may not be suitable for many other people and is, in any case, only a partial one.

Arrays are programming constructs that confuse some people and, for beginning students, take some getting used to.
students, take some getting used to. Instead of providing a gentle introduction, reminding people that lists are like arrays may have just the opposite discouraging effect because it reminds them of previous difficulties. “Great!”, someone might think. “To learn a complicated new subject, I have to relate it to something else I had trouble learning.” Instead of comparing lists with arrays, why not compare them with SAS data sets and flat files, a reference that is non-threatening and also may lead toward more rapid understanding?

Let us assume that someone attempting to learn SCL lists knows base SAS software and is comfortable working with SAS data sets. What does the person already know that is relevant? Here are two simple programs that will provide us with an answer:

```
DATA ONE;
  X:::1;
RUN;
```

A person knows this program creates a structure that holds data, a SAS data set named ONE, with one observation and one variable, X, whose value is 1. The person also knows that the assignment statement populated the structure automatically. Here’s another program:

```
DATA TWO;
  DO i = 1 to 20;
    X:::i;
    OUTPUT;
  END;
RUN;
```

The person also knows this program creates a structure that holds data, a SAS data set named TWO, with 20 observations containing incremented values of X. But most of all, people already know something about general behavioral characteristics of data structures in SAS. They know that a SAS data set (data set ONE) can be filled with values automatically. The assignment statement “X=1” illustrates that behavior. Second, they know that the size of a SAS data set (data set TWO in this case) can grow dynamically.

Based on this general understanding, they can realize that SCL lists sometimes behave the same way. The DATALISTC function in SCL, for example, populates a list automatically and dynamically.

SAS data structures have other behavioral characteristics. Here is a different simple illustration:

```
DATA _NULL_;  
  FILE PRINT;  
  SET SASDS;  
  PUT @1 NAME $10.  
      @i5AGE3.;  
RUN;
```

Everyone feels comfortable with this kind of program, even though they may not like the tediousness of writing DATA _NULL_ code. What experience can they draw on? They know it’s necessary to specify each thing they want to put into another data structure called a file, for example the exact place it must go. People understand that files, unlike data sets, must be spoon fed. Lists sometimes can be filled with data in the same way. The INSERTC function in SCL, for example, is a cousin of the PUT statement in SAS. As in:

```
listid=insertc(listid,'CHICAGO',3);  
```

There’s nothing automatic here.

Conversely, people know that reading data in from a flat file necessitates providing detailed specifications about the data. This kind of painstaking code
is similar to the SCL functions that retrieve items from lists.

What else does the DATA _NULL_ program reveal? That when writing to a flat file, one can mix character and numeric variables. Although mixing the two data types is often mentioned as a hallmark of new list technology, SAS programmers have been doing this for years with flat files, ever since they first learned SAS. Why not draw on these experiences, these frames of reference?

It isn't necessary to go beyond this level of code to realize what may needed be to improve learning about lists. Lists combine features of data sets and flat files (and arrays too, of course). Lists can be driven either with automatic or manual transmission. This characterization isn't obvious if list functions are presented alphabetically; the multifaceted behavior of lists that resembles the behavior of other SAS data structures is hidden.

SUBCLASS

The introduction of Frames and the use of object-oriented constructs in the SAS system have opened up exciting prospects for system developers while posing new challenges for learning. It's normal that new and powerful techniques - in this case object-oriented programming - have considerable appeal.

Object-oriented programming has its own terminology, nouns such as class, inheritance, instance, and verbs such as override and subclass. People may attempt to learn this terminology on its own terms. However, just as with lists, it may be helpful to understand parallels with other SAS system software when learning this new technology.

The word "subclass" was suggested to the author as an example of a troublesome concept. In object-oriented glossaries, this term is related to the word "class" which in the SAS system, is defined as "the template or model for an object". If that is a class, then what is a subclass? Does this definition really hit the nail on the head for everyone?

The previous discussion of lists attempted to show that people with a knowledge of SAS data sets and flat files already have a basic understanding of things that can be applied to Screen Control Language. Instead of fighting to decipher something mysterious and powerful, they can unlock its mysteries with a key.

Look at another program that works with the data set created in the previous section. Everyone can understand the following:

```
PROC PRINT DATA=ONE;
VAR X;
TITLE "Jim's Dazzling SAS PROC PRINT";
RUN;
```

This program illustrates use of a simple SAS procedure. Everyone is familiar with at least a few procedures. They know that the syntax of PROC FREQ differs from PROC PRINT, that some procedures are more complicated than PROC PRINT. However, regardless of their variety and complexity, procedures have certain characteristics in common. For example, they have certain default behaviors that may be modified through selective use of their options.

In the above example, the optional TITLE statement replaces the default title ("The SAS System") with one written by the programmer. Note that while the procedure allows optional use of procedure enhancements, it does not
allow custom, user-written enhancements. For example, if one wanted to left-justify the title, there is no provision to make a clone of the PRINT procedure and modify it with a new option - let's call it 'L TITLE' for left-justified title. Such a new procedure would look like this:

```sas
PROC PRINT DATA=ONE;
   VAR X;
   LTITLE "Jim's Dazzling New Clone Of PROC PRINT";
RUN;
```

People can understand the theoretical PROC PRINT variation presented here. Unfortunately, in the world of procedures, such a feat is only imaginary. However, magic like this is possible in the realm of object-oriented programming. There, a procedure is a 'class' and options are 'methods'. The clone of the class is a 'subclass'. To subclass is to clone. All that someone needs to learn is the mechanics.

Procedures and classes only look different but they are still related, as are line printer and full color graphics. Object-oriented programming did not arise out of thin air and the procedural antecedents of classes in SAS programming are worth considering.

**Terminology and Understanding - Toward More Expressive Language**

**EXTENDED TABLES**

As we have just seen, one can accept terms like 'list' and 'subclass' as reasonably descriptive and find meaning through analogies with easily understood base SAS software. Sometimes, terminology is the issue. In the case of extended tables, it is more helpful to pay close attention to language and meaning than in the cases of lists and subclasses. Here, the key to understanding ultimately requires analyzing the subject matter on its own, rather than comparative, terms. Admittedly, this may be more demanding. There are no simple, familiar analogies with base SAS software to speed understanding.

To be sure, it might be worth analyzing the expression 'extended table' itself, since it isn't intuitive. Instead, let us for the moment live with it and see how terminology within extended tables is somewhat misleading. The following discussion illustrates that the need to improve the learning process doesn't stop. It also applies to more advanced levels of learning.

An extended table displays data - usually from a SAS data set - in a scrollable form. The syntax of Screen Control Language for extended tables is broken into several sections whose statements manage different characteristics of this scrollable display: 1) how many rows to display, 2) filling the scrollable display, 3) modifying or selecting the rows, and 4) ending the display.

A common use for an extended table is to display values as a selection list. One positions the cursor on a particular row of the list, presses the ENTER key, and then something happens based on the row selected. For example, if a list of names is displayed and one of them selected, a report can be printed for the name selected.

The functionality of the extended table seems to be straightforward. A SETROW statement lets you control how big the table is - i.e., how many names will be displayed, a GETROW section displays the names, and a
PUTROW section triggers some action when a name is selected.

The SETROW statement has some slightly unusual syntax, but other than that is reasonable descriptive. The word ‘GETROW’ also has unusual syntax and seems clear because it acts to get what is displayed. The word ‘PUTROW’ also is more or less reasonable since action is taken on the selected row - it was ‘put’ somewhere, one imagines.

Why, then, is there a problem with terminology? In a way, the problem is psychological. People want to learn about extended tables so they can do something with them. What more obvious use can there be than to take action on a selected row? Therefore, from the beginning, one learns that the meatiest portion of extended tables is the PUTROW section. This is what they’re good for. This is where the action is. And the distraction. From what?

We are dealing here with something mysterious. One of the other characters in extended tables isn’t the innocuous statement or section label it pretends to be. It isn’t just a supporting character. One of the characters is in disguise and people don’t recognize it. In fact, the problem is that the disguise is so good that no one realizes there is a mystery at all.

There are few clues to be found in any documentation this observer has seen. It was only by spending a night in the house of extended tables that the disguised character revealed itself.

Here is some simple code:

```
INIT:
dictid=open('sample.dataset','i');
call set(dictid);
umobs=atlrn(dictid,'nobs');
call setrow(numobs,1); RETURN;
```

The first time the program runs the input data set is opened. The data are aligned with screen variables and the number of rows in the table is determined.

```
GETROW:
rc=fetchobs(dictid,_currow_);
RETURN;
```

This section fills the rows. From here the program could continue with a PUTROW section - because that is where the action is supposed to happen. But suppose we go back and do the following:

```
GETROW:
rc=fetchobs(dictid,_currow_);
shownum= varnum(dictid,'showrep');
repready= getvarc(dictid,shownum);
if repready='N' then
do;
  report = report II ' (NY A) ';
end;
RETURN;
```

Something different happens now. As the table’s rows fill up, each row may have a special action taken on it, based on the values of the input data. In this example, each report in a list of report titles may appear either as is, or with the notation ‘NYA’ (not yet available) after it. The GETROW section now does more than just fill the table. It evaluates the input data record by record and takes
The mystery is solved. GETROW doesn't 'get a row'! GETROW's real function is to initialize the scrollable display and, can, if desired, do this initialization as powerfully - using SCL - as any other kind of INIT section in Screen Control Language! Think of the possibilities if rows can be displayed conditionally. Other kinds of action are possible. For example, after one row is selected and action taken in PUTROW, the next time the table is displayed it can have other characteristics than before PUTROW acted.

This makes it possible to develop very sophisticated applications. However, because GETROW is in disguise, this fact is not immediately apparent. Understanding the function of GETROW opens the door to using extended tables in more advanced applications. That is another way to expand the use of the SAS system.

What should be GETROW's real name? Shouldn't it be something more informative?

CONCLUSION

This paper has gone on a journey into the subjective side of learning SAS software. It has explored ways to uncover meaning in different ways, first by looking for easy-to-understand analogies where that method is appropriate and then by looking at function before terminology. It has tried to draw attention to the subjective side of learning and to identify some possible ways to improve the learning process. It has done so in the belief that the subjective is the strategic.

The paper contends that this is a necessary exploration, and it is hoped there will be more. What has begun to be mapped out here needs considerably more extensive cartography.

REFERENCES

1. Another, somewhat vague, comparison is that they are ordered collections of data.
2. It is possible to see other analogies with data sets. For example, descriptive values such as 'COUNT' can be retrieved from lists just as one can retrieve descriptive information about SAS data sets using PROC CONTENTS.
3. When the SAS MACRO language came out, many programmers rushed to convert all their code into macros.
5. PROC SHEWART in SAS/QC has considerably more than 200 options.

SAS, SAS/AF, and SAS/QC are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries.® indicates USA registration.

Other brand or product names are registered trademarks or trademarks of their respective companies.

AUTHOR

Dr. Jim Sattler
Saber Consulting Services, Inc.
3888 Balcom Road
San Jose, CA 95148
Phone/Fax>408.270.4670
email> jsattler@saberconsult.com