Introduction:

Macros called SET and MERGE can be used to replace the SAS® commands of the same names to make life easier for users of SAS programs. At the U.S. Army Aviation Systems Command (AVSCOM) in St. Louis, Missouri, the Sample Data Collection (SDC) office collects data on a sample of approximately 850 Army aircraft. The SDC data base stores data regarding the maintenance and operation of these aircraft. The data can be used to support engineering studies. The focus of the SDC office has been to provide a data base of SAS files and a library of usable SAS programs to the AVSCOM engineering community. Although we provide printed reports on request, the focus has been to provide the tools for the engineer or analyst to conduct studies using the data base in a dynamic fashion.

Using Macro Variables in SDC Programs:

Standard SDC programs are set up with a set of macro variables at the start of each program. By changing the settings on these macro variables, the user can produce numerous different listings from the same basic program. The macro statements most used are:

```
%LET AIRCRAFT = AH64A;
%LET STDATE = 01JAN84
%LET ENDATE = 31DEC88
%LET SELECT = LOC = 'HO' and LEVEL = 'U'
%LET SORTORDR = WUC_FG
```

In the first statement, the user supplies the name of the aircraft he or she wishes to run the program for. Two examples are 'AH64A', which is the Apache helicopter, and 'UH60A', which is the Blackhawk helicopter. In the second and third statements, the user supplies a start date and an end date to define the period of time the run will cover. In the fourth statement, the user defines the macro variable SELECT. An example is: LOC = 'HO' and LEVEL = 'U'. This indicates that the user wishes to select only records from Ft. Hood and only records involving the unit at the AVUM maintenance level. This is the aviation unit maintenance level, that is, the level where maintenance is done by the unit which owns the aircraft. For the SORTORDR macro variable, the user selects the order that the listing will be printed in. For example, %LET SORTORDR = WUC_FG; WUC_FG stands for 'Work Unit Code Functional Group' and is a classification of the maintenance event by the part of the aircraft serviced. For example, '05' indicates the rotor system on the Apache. This is not a natural language system, since the user must refer to a code book to specify his SELECT and SORTORDR. However, this system does allow the user to produce many types of listings without making any changes to the program code. The programmer has embedded in the program an 'IF &SELECT;' and a 'BY &SORTORDR;' at the appropriate places in the program. This system has been very useful.

File Organization:

The SAS data sets are organized so that there are several files for each weapon system or aircraft and these files are organized together into a library identified by the aircraft name. Hence, for the Apache (AH64A), the SAS data sets are AH64A.EV10, AH64A.EV20, AH64A.EV22, etc. When the user types

```
%LET AIRCRAFT = AH64A;
```

a statement in the program

```
SET &AIRCRAFT..EV30;
```

and another statement

```
SET &AIRCRAFT..EV35;
```

will automatically pull in the appropriate data sets. However, the structure of the data sets did cause a problem here.

Over the years, a large amount of data has been collected. It is very inefficient for SAS to read data from 1977 to 1989 on every run when users will select data from only the last two years. Hence, the files have been divided into archived and current data files. The year 1988 is our current break year. The archived data are placed in a SAS library called by the aircraft name attached to the characters 'B88'. For example, AH64AB88.EV20. The current data are stored without the 'B88'. For example, AH64A.EV20. Due to this file structure, certain changes must be made to the code depending on the dates selected. For example, if a user selects STDATE = 01JAN86 and ENDATE = 31DEC88, then data steps such as the following will be necessary:

```
DATA TEMP;
SET &AIRCRAFT..B88.EV30 &AIRCRAFT..EV30;
BY CNTLNUM;
IF &SELECT;
RUN;
```

On the other hand, if a user selects STDATE = 01JAN84 and ENDATE = 31DEC88, then the following code is appropriate:

```
DATA TEMP;
SET &AIRCRAFT..EV30;
IF &SELECT;
RUN;
```

Again, if a user selects STDATE = 01JAN84 and ENDATE = 31DEC88, then the following code is needed:

```
DATA TEMP;
SET &AIRCRAFT..B88.EV30;
IF &SELECT;
RUN;
```

A technique was needed to avoid making changes to the code to accommodate changes in the dates.

Prior to 1988, an SDC program user who wanted to read both current and archived data was required to run a "COMBINE" program which would indeed combine the current and archived data sets. The user then ran the SDC program against this combined set. This required a large amount of temporary disk space to store the combined data sets while they were being used, but it did obviate the necessity to rewrite the code. The combined sets were just renamed AH64A.EV20; or whatever name was needed. Another drawback to this method was that it required the data sets to be read twice. They had to be read once to combine them and then read once again in the actual program. "PROC APPEND" was not an option, since the data sets had to be in control number order for processing and the B88 data sets have some control numbers which are higher than some control numbers in the current data sets. Hence a "SET" statement and a "BY CNTLNUM" statement had to be used which meant a complete read through prior to starting the SDC program.

The SDC office has investigated storing data in System 2000® or some other type of data base system and then using the SAS interface to bring the data up as SAS data sets. This was not an option for us at the beginning of 1988 and is still remote to us as an option. However, we did discover that we could make use of a "SET" macro and a "MERGE" macro to replace the SET and MERGE commands which would allow us to eliminate the "COMBINE" program. I will discuss the SET macro because the MERGE macro works essentially in the same way.
SET Macro:

The SET macro works as follows. Whenever any program in the SDC library encounters a SET command, this command is treated as a statement type macro. In other words, it is recognized as a macro even though no percent sign cues the interpreter that SET is a macro. The option "IMPLMAC" is used so that the first word of each command will be tested to see if it is a statement macro. Under the CMS operating system, we store this SET macro in a MACLIB which we reference in our SAS profile. Hence the macro is always available to the program.

The SET macro works essentially by checking the start and end dates specified for the run and then specifying the correct data sets after the word SET. A "BY CNTLNUM;" is added if more than one data set must be specified. Before the macro can rewrite the SET command, however, it must read all the options on the SET command. For example the "END = EOF" options or the "RENAME=" option. The SET macro uses the PBUFF option to pick up all the words following the word 'SET' on the SET command. It then issues the command

```
LET WORDS = &SYSBUFF;
```

The macro variable WORDS is then examined to parse it into data sets and options.

The steps of the parsing operation are as follows:

1. Define as the macro variable WORDS everything after SET.
2. Examine WORDS for the presence of parentheses. Replace outer parentheses by "." We now have:

```
WORDS:
```

3. Parse WORDS into phrases using the %QSCAN function, scanning for "."
4. Define a type for each phrase. The type will be either DSNAME or PAREN.

The type is defined by alternation. The first type will always be DSNAME representing one or more data set names. The next phrase will always be of type PAREN.
5. If the type is DSNAME, then the phrase is further parsed into subphrases by scanning for blanks. The resulting subphrases are data set names or a "LAST" phrase. The "LAST" subphrase is recognized by the presence of an "=" either in that subphrase or in the next subphrase, as in END=EOF or END = EOF. Note that this step usually requires one look ahead of the next subphrase to be sure a "LAST" subphrase is not mistaken for a data set name.
6. Upon recognizing a subphrase as a data set name, the macro then holds this name until a PAREN type phrase can be read in. When the macro recognizes that both a data set name and a PAREN type phrase have been read in, then it will rewrite these words on the SET statement. If the held phrase and the next phrase are both data sets then the macro will go ahead and write just the data set and put the second data set on hold.
7. When the macro rewrites the data set name, it uses a macro variable called "TASK". This macro variable is defined earlier in the program and tells the SET macro how to rewrite the data set name. For example, if the task if "COMBINE" then the macro replaces AH64A.EV30 with AH64AB88.EV30. If the task is "RENAME", then the macro replaces AH64A.EV30 with AH64AB88.EV30. The task "COMBINE" causes the program to read both archived and current data. The task "RENAME" causes the program to read only archived data. The "TASK" macro variable is defined by examining the STDERR and ENDATE macro variables. The variable "TASK" is only considered when an SDC data set is under consideration. These data sets are recognized by checking whether the first part of a two-part name is &AIRCRAFT. Hence the data set name ADDON is not affected. If "LAST" exists, it is added to the end of the new SET statement. If a "BY CNTLNUM;" is needed it is also added. Suppose "TASK" is "COMBINE". Then the new SET statement looks like this:

```
SET AH64A.EV30 (DROP = WUC NOMEN RENAME = (PART = PART30)) AH64AB88.EV30 (DROP = WUC NOMEN RENAME = (PART = PART30)) ADDON (IN = IN1) END = EOF;
```

Other Uses For The SET and MERGE Macros:

Since implementing this system, we have discovered that having specialized SET and MERGE macros has made certain other enhancements to our system possible. For example, we now have an option on our %LET SELECT statement to allow the program to run against a selected set of records. The control numbers of these records are placed In a data set and then that data set is identified in the SELECT statement. For example, suppose our SAS program has identified the control numbers of a selected set of records which need further analysis by another program. The first program places those control numbers in a file called BAD.RECS. The second program is run by setting %LET SELECT = CONTROL BAD.RECS; A special SELECT processor recognizes the word CONTROL in the SELECT macro variable. It then sorts the records in BAD.REC by control number, sorts a flag, and renames BAD.REC to CONTROL. When SET sees the flag, it converts the SET statement to a merge as follows:

Original:
```
SET AH64A.EV30;
```

Converted:
```
MERGE AH64A.EV30 CONTROL (IN = INCNTL); BY CNTLNUM;
```

Another use of this CONTROL data set is in allowing selects on outside data sets. For example, if the SELECT processor recognizes certain key words, such as "FTX", which is an acronym for Field Training Exercises, it creates a data set CONTROL of just FTX events. It then merges those events into the original data set as described above.