The Painless Path to Release 6.06 of the SAS® System

Robina G. Thornton, SAS Consulting Services Inc., Rockville, MD
John Boling, SAS Institute Inc., Cary, NC

ABSTRACT

This paper addresses two main issues. The required changes for application conversion from Version 5 of the SAS® System to Release 6.06 is addressed first. Next, selected new features available in Release 6.06 of the SAS System that are not required for application conversion but can enhance the application in terms of efficiency, maintainability, or required development time are discussed.

INTRODUCTION

The paper provides a general overview of the enhancements to the base SAS software product without regard to mode of execution (that is, batch, interactive). Some of the features to be discussed include indexing SAS data files, compressing SAS data files, storing compiled DATA step code, creating VIEWS using the SQL procedure, and WHERE processing. For more detailed information on the features discussed in this paper the reader should consult the appropriate documentation. Throughout the paper examples are presented using data for a large manufacturing company. Most of the examples reference either the employee master file or the ordering/invoicing data.

THE SAS DATA MODEL

The internal structure of SAS files has changed in Release 6.06 of the SAS System, which allows for new concepts for the storage and access of data by the SAS System. These new concepts expand the functionality of the SAS System without greatly changing the everyday use of the SAS language. As always a SAS data set consists of two parts: a descriptor portion that describes the attributes of the variables in the data set, and a data portion containing the actual data values. In Release 6.06 a SAS data set is called a SAS data file (memtype=DATA) if the descriptor portion and the data values are in the same physical location. If the descriptor information and the data values are stored separately, they form a SAS data view (memtype=VIEW). The view descriptor includes information as to where the data are physically stored and which records and variables to process. Some people prefer to think of this data view as a logical SAS data set. Within the SAS System both SAS data files and SAS data views are referred to as SAS data sets. The data values for a SAS data view can be stored in a SAS data file, an external data base, or an external file. Therefore, the SAS System can directly access data from any of the following sources as if the data were contained in a physical SAS data file without ever creating the physical SAS data file:

- DBMSs such as DB2®, ORACLE®, Rdb/VMS®, SQL/DS®, and SYSTEM 2000® (using SAS/ACCESS® software)
- data created by statistical software products such as BMDP, SPSS, and OSIRIS
- logical data sets (VIEWS) of variables stored in various data files including SAS data sets.

CONVERSION REQUIREMENTS

For many applications nothing will need to be changed to run an existing application under the new release of the software. Although the file structure within the SAS System has changed, this change has little impact on the compatibility of existing Version 5 SAS programs with Release 6.06. Unfortunately, there is no simple rule that applies to every existing application. Exactly what and how much needs to change in current production applications depends on the application and the operating system. The following sections discuss three different scenarios.

Base SAS Applications

If the application is a batch base SAS application using only Institute supported procedures (that is, no procedures from the Supplemental Library or other user written procedures that have not been absorbed into the base product) the application should run under Release 6.06 with no changes. The new Multiple Engine Architecture enables you to access a variety of different types of files as if they were SAS data sets. There are several native SAS engines that are contained in the base SAS software product. One of these engines is the V5 engine, which allows the program running under Release 6.06 to access Version 5 SAS data sets. When a Version 5 library or SAS data set is the source of input to a Release 6.06 step, the V5 engine is engaged to process the input data, and the SAS System will then generate the output data set or library using the Release 6.06 architecture. The fact that the input data files are of a different format is transparent to you.

If conversion of SAS data sets is necessary or desirable you may use a DATA step, the new COPY statement of the DATASETS procedure, the COPY procedure, or the new V5TOV6 procedure. The new procedure converts SAS data libraries (all member types) to the new Release 6.06 architecture on the same operating system. The following example demonstrates the use of PROC V5TOV6:

```
LIBNAME V5DATA 'SASID.EMPLOYEE.LIB';     /* version 5 library */

LIBNAME V6DATA 'SASID.EMPLOYEE.LIBV6';  /* version 6 library */

PROC V5TOV6 IN = V5DATA  OUT = V6DATA;
RUN;
```

Applications Utilizing User-Written Formats or Informats

In Release 6.06 user defined formats and informats are stored in SAS libraries and have the member type of CATALOG. If an application accesses user created formats and is running in the minicomputer environment, the format library must be converted to a Release 6.06 catalog before the application can be run under Release 6.06 software. To convert formats and informats from Version 5 to Release 6.06 in the minicomputer environment you MUST use PROC V5TOV6 as in the following example:

```
LIBNAME V6DATA 'SASID.EMPLOYEE.V6LIB';   /* version 6 library */
LIBNAME FMTLIB 'SASID.EMPLOYEE.FMTLIB';  /* version 5 format lib */

PROC V5TOV6 FORMAT = FMTLIB OUT = V6DATA;
RUN;
```

In the mainframe environment (MVS, CMS, or VSE) the conversion is not required, but if conversion is desired the PROC must use the FORMAT = SASLIB option where SASLIB is the mandatory libref of the Version 5 format library.

Applications Utilizing SAS/AF® Software or SAS/FSP® Software Catalogs

If an application uses SAS/AF software or SAS/FSP software catalogs, conversion of the catalog using PROC V5TOV6 is required. If you convert a catalog that contains entries of type PROGRAM, you must compile the entries before execution. Version 5 SAS/AF PROGRAMS will be converted by the V5TOV6 procedure to Screen Control Language (SCL) programs in your SAS/AF applications. The conversion the SAS System will perform is straightforward. Once you have converted the catalog using PROC V5TOV6, the application will run without further changes, although you will most likely want to enhance and modify the SAS System generated SCL to take full advantage of the power of SCL in your SAS/AF applications. Triple pound sign (###) macros will not be converted to SCL. Instead, each triple pound sign macro is converted to a separate catalog entry of the type AFMACRO. Each entry's name is the name of the original macro.

The following table indicates what needs to be converted:

Version 5 file	Conversion	Release 6.06 Memtype
data sets	optional	data
formats/informats	optional (mainframe) required (mini)	catalog
SAS/GRAPH® catalogs	required	catalog
SAS/FSP catalogs	required	catalog
SAS/AF catalogs	required	catalog
SAS/IML® modules	required	catalog
matrices	required	catalog
SAS/ETS® models	required	catalog

A Version 5 data library could include SAS data sets, SAS/AF catalogs, SAS/FSP catalogs, SAS/GRAPH catalogs, SAS/ETS models, and SAS/IML matrices and modules. One execution of the V5TOV6 procedure can perform the required conversion for all these members of the library. The only SAS created file entities that cannot be converted and must be re-created are SAS/GRAPH font libraries.

SELECTED NEW CAPABILITIES

There is new functionality available in the software as well. These are features you may want to take advantage of not only when writing new applications but also when upgrading Version 5 applications. These features are *not* required to run a Version 5 application under Release 6.06, but depending on your existing application, incorporating these features may improve the efficiency and/or maintainability of your existing application.

IN Operator

The IN operator is a new comparison operator that facilitates making comparisons to a list of items. It is similar in use to the IF statement. In Version 5, to process a subset of all the employees under the vice-president of finance some users may have written the following:

```
DATA FINANCE;
  SET   CORPDATA.MASTER;
  IF DEPT = 'ACCT' OR
     DEPT = 'PAYROLL' OR
     DEPT = 'LEGAL' OR
     DEPT = 'INVOICE' OR
     DEPT = 'ORDER';
   other processing code here
RUN;
```

Now in Release 6.06 the code may be written as

```
DATA FINANCE;
  SET CORPDATA.MASTER;
  IF DEPT IN ('ACCT', 'PAYROLL', 'LEGAL', 'INVOICE', 'ORDER');
   other processing code here
RUN;
```

The IN operator expects character values in quotes and the values separated by commas or blanks. The IF statement is still present in Release 6.06, so code does not need to be updated. However, you may want to take advantage of the conciseness of the new supported syntax.

WHERE PROCESSING

WHERE processing is now available as both a data set option and a statement in both DATA and PROC steps. WHERE processing allows you to select a subset of observations satisfying one or more conditions in an existing SAS data set. An example of a WHERE statement to select employees from the accounting department follows:

```
LIBNAME CORPDATA 'SASID.EMPLOYEE.LIBV6'; /* version 6 SAS library */
DATA ACCTNG;
   SET CORPDATA.EMPLOYEE;
   WHERE DEPT = 'ACCT';
   other processing statements
RUN;
```

An example of a WHERE data set option follows:

```
LIBNAME CORPDATA 'SASID.EMPLOYEE.LIBV6'; /* version 6 SAS library */
DATA EMPLOYEE;
   SET CORPDATA.EMPLOYEE ( WHERE = (DEPT = 'ACCT'));
   other processing statements
RUN;
```

Or the WHERE processing can be in a PROC step as either a statement or data set option

```
LIBNAME CORPDATA 'SASID.EMPLOYEE.LIBV6'; /* version 6 SAS library */
PROC PRINT DATA=CORPDATA.EMPLOYEE ( WHERE = (DEPT = 'ACCT'));
RUN;

PROC FREQ DATA=CORPDATA.EMPLOYEE;
   WHERE DEPT = 'ACCT';
   TABLES SALARY;
   FORMAT SALARY SALFMT.;
RUN;
```

The expression can be any valid arithmetic or logical expression. The expression generally consists of a sequence of operands and operators. The operands can include constants, values of variables, and values created within the WHERE expression. The operators can be logical, comparison, arithmetic, IN, or special WHERE expressions. The following five new operators are valid in a WHERE statement:

- BETWEEN - AND to select observations based on a range of values

```
WHERE SALARY BETWEEN 20000 AND 25000;
```

- CONTAINS or ? to select observations that contain the character string specified

```
WHERE EMPNAME CONTAINS 'STR';
WHERE EMPNAME ? 'TON';
```

- IS NULL or IS MISSING to select observations for which the variable value is missing or null

```
WHERE STARTDT IS MISSING;
```

- = * (sounds like operator) to select observations that contain a spelling variation of the word specified

```
WHERE FRSTNAME = * 'ALAN'
```

which would also find 'ALLAN' and 'ALLEN', and so on.

- LIKE to select observations with character values matching a specified pattern. To specify patterns, there are two special characters available:

% (percent)
represents any number of characters in that position

_ (underscore)
represents one character in that position

```
WHERE EMPNAME LIKE 'S%'
```

to select all employee names beginning with the letter S or

```
WHERE EMPNAME LIKE 'S_E%'
```

to select all employee names starting with the letter S with a third letter of E.

WHERE processing can be used to select or subset a data set. It is not always a substitute for a subsetting IF statement. The WHERE statement or data set option

- can only be used with existing SAS data sets
- is not executable
- selects observations before they are copied into the program data vector
- applies selection criteria to each input data set in a MERGE statement before combining the current observations
- can produce results that are different from a subsetting IF in steps that interleave, merge or update data sets
- does not allow the use of functions in the expression.

A major advantage of the use of WHERE processing with a PROC step is that several separate steps can be eliminated in many cases. Take a situation that requires a subset of the data, followed by a sort of the data, followed by a print step. In Version 5 that required three separate steps and three passes of the data, as illustrated below:

```
DATA MALES;
   SET CORPDATA.EMPLOYEE;
   IF SEX = 'M';
RUN;

PROC SORT DATA = MALES;
   BY DEPT LASTNAME;
RUN;

PROC PRINT DATA = MALES;
   BY DEPT;
   SUM SALARY;
RUN;
```

Now code two steps with a WHERE statement in the PROC SORT as follows:

```
PROC SORT DATA = CORPDATA.EMPLOYEE OUT = MALES;
   WHERE SEX = 'M';
   BY DEPT LASTNAME;
RUN;

PROC PRINT DATA = MALES;
   BY DEPT;
   SUM SALARY;
RUN;
```

Additionally, a WHERE command is available in SAS/FSP software that subsets the data as well. While a WHERE command is in effect the word *where* appears at the top of the FSEDIT screen as a reminder. If a WHERE command is already active, use a WHERE ALSO to further subset the data. These commands can be nested to any level desired. To back out of the nested WHERE commands use the WHERE UNDO command, and you are backed out one level at a time. To eliminate all the WHERE subsetting that is in effect, use the WHERE CLEAR command.

Stored Program Facility

The new stored program facility allows you to store compiled DATA steps for later execution. This can save time at execution in large production runs. The stored DATA steps can contain any valid DATA step statements excluding global statements, such as LIBNAME, FILENAME, TITLE, FOOTNOTE, or OPTIONS, and host-specific data set options or host-specific FILE and INFILE statement options. Use of the facility is a two part process: first, the DATA step is compiled and stored and second, the compiled code is executed redirecting the input and output as necessary. The SAS System does not save the source code and it cannot restore source code from the compiled version, so it is necessary to permanently save the source statements. To compile and store a DATA step, submit the step using the PGM option on the RUN, CARDS, or CARDS4 statement that ends the DATA step. Using the employee file, create a new data set that contains all the employees who have resigned so that the appropriate paper work will be produced. This program will run every Friday, and the DATA step needs to be stored using the stored program facility. The first step is to code and test the DATA step. Once the step has been tested, store the compiled code as follows:

```
DATA  CORPDATA.RETIREE;
  SET CORPDATA.EMPLOYEE;
  IF (TODAY() - 6 )  LE ENDDATE LE TODAY();
RUN PGM = CORPDATA.RETPGM;
```

The member RETPGM is stored in the SAS data library with a memtype of PROGRAM and can now be called at execution time. It is important to note that the compiled program cannot run under a different release of the SAS System. To execute the compiled step, an abbreviated DATA step must be written using the PGM option on the DATA statement, such as

```
DATA PGM=CORPDATA.RETPGM;
RUN;
```

SQL Procedure

The Structured Query Language is a standardized high-level query language that is widely used to retrieve and update data in relational database management systems. In Release 6.06 the new procedure PROC SQL uses the structured query language to perform the following functions:

- retrieve and manipulate information stored in SAS data files, PROC SQL views, and SAS/ACCESS views
- create and delete data sets, views, and indexes
- generate reports
- add or modify the values in a data set
- add, modify, or drop varaibles in a data set.

In the SAS System a SQL table is a SAS data set. This procedure is extremely powerful and can accomplish the work of many traditional SAS steps in one procedure. The SELECT statement has several clauses and can be used to select data from one or more data sets that meet stated criteria, group or sort the data, create calcu-

lated fields, and process the data in specified groups. The first PROC SQL example demonstrates the creation of a PROC SQL view. The view created is a subset of the variables and records from the employee master SAS data file. This view does not require the physical storage space that a SAS data file requires because the actual data values are not stored in the view. The view contains information to the SAS System on how to locate the appropriate data values. Also, when PROC SQL executes, the SAS System retrieves the data values at execution time, thus always accessing the most up-to-date version of the data. Note that PROC SQL views have read-only access. In an environment where subsets of large files are created on a regular basis for users to access their data, PROC SQL can reduce both execution time and storage requirements.

```
PROC SQL;
   CREATE VIEW CORPDATA.REVIEW AS
   SELECT EMPNAME, STARTDT, SALARY, BONUS88, BONUS89, STATUS,
          TITLE, REVDATE
   FROM CORPDATA.EMPLOYEE
   WHERE (REVDATE BETWEEN 01APR90 AND 30APR90) AND (DEPT='MARKET')
   ORDER BY EMPNAME;
RUN;
```

The SQL code creates a permanent SQL view named REVIEW. It contains the variables in the SELECT statement and the marketing department records for the scheduled April performance review. The data are sorted alphabetically by employee name. This view can now be accessed as a permanent data set, and at execution time the SAS System will retrieve the actual data values, obtaining the most current data. The manager of this department can now perform any necessary data analysis on "his" subset of the employee master file.

In the second example for PROC SQL, a comparison is made of several steps using Version 5 (taken from *SAS Views®: SAS Processing*) and the one PROC SQL step that can now be written in Release 6.06. Both programs illustrate the combining of data from three separate files, ORDERS, PRICES, and CUSTOMER, to produce bills for the regular customers.

```
PROC SORT DATA=CORPDATA.ORDERS OUT=ORDERS;
   BY PRODUCT;
RUN;
DATA ORDERED;
   MERGE ORDERS (IN=ORD) CORPDATA.PRICES;
   IF ORD;
RUN;
PROC SORT DATA=ORDERED;
   BY CUSTOMER;
RUN;
DATA INVOICES (KEEP = CUSTOMER CITY STATE COMPANY TOTAL);
   MERGE ORDERED (IN=ORD) CORPDATA.CUSTOMER;
   BY CUSTOMER;
   IF ORD;
   IF FIRST.CUSTOMER THEN TOTAL = 0;
   TOTAL + (QUANTITY * UNITCOST);
   IF LAST.CUSTOMER;
RUN;
PROC PRINT DATA = INVOICES;
RUN;
```

The PROC SQL code produces a report with the same information.

```
PROC SQL;
   SELECT DISTINCT CUSTOMER.CUSTOMER,
          CITY,STATE,COMPANY,
          SUM (QUANTITY * UNITCOST) AS TOTAL
   FROM CORPDATA.CUSTOMER, CORPDATA.PRICES, CORPDATA.ORDERS
   WHERE CUSTOMER.CUSTOMER = ORDERS.CUSTOMER AND
         PRICES.PRODUCT = ORDERS.PRODUCT
   GROUP BY CUSTOMER
   ORDER BY CUSTOMER;
RUN;
```

Indexes

Another feature available with Release 6.06 is that SAS data files (Version 6 data files only) can be indexed by one or more variable. There are two major benefits when using an index: fast access to a small subset of records and data retrieval in order of the index without using the SORT procedure. Consider the use of FSEDIT to allow data entry in Version 5. To ascertain the presence of duplicate records for a unique key, several steps are involved. First the user enters the data via the FSEDIT procedure. Then the SORT procedure is run based on the unique key. Next, a DATA step must be executed to perform FIRST. and LAST. processing to eliminate the duplicates. Depending on the application, a printout of the duplicates may be desired. Now the use of indexes eliminates much of this processing. If the file being edited with PROC FSEDIT can be indexed using the UNIQUE option on the keys, the procedure will not allow the user to enter records with a duplicate key value. As can be seen in the following example, another benefit is that files can be processed without the resources required by PROC SORT. Refer to the examples used in the discussion of WHERE processing. If the file were indexed by DEPT and NAME PROC SORT is eliminated because the WHERE option retrieves the values in index order.

```
/* produce listing sorted by DEPT and LASTNAME for MALES only */
/* sum the SALARY for each DEPT                               */

PROC PRINT DATA = EMPLOYEE (WHERE =(SEX='M'));
   BY DEPT;
   SUM SALARY;
RUN;
```

An index stores the values of SAS data set variables and a system of pointers that enable the SAS System, under certain circumstances, to locate observations in the SAS data set more quickly and efficiently. Once you create an index, the SAS System determines when to use it. The index is stored by the SAS System as an inverted tree structure and is automatically maintained by the SAS System. The index file does not appear as a separate SAS file within the SAS environment, but will appear as a separate SAS file within the operating system environment, except under MVS. Indexes should only be deleted by the SAS System.

There are two index structure types, a regular or single key index and a composite or multikey index in which the several values are concatenated to form a single value. One data set can have several regular and composite indexes. Indexes can specify that the index value is unique, only allowing one record for each value of the index. The index attribute can also specify that missing values are not acceptable in the index file but are allowed as the value of the index variable. When an index is created you specify the index name (a valid SAS name), the attributes, and whether it is regular or composite. There are several techniques for creating an Index for SAS data sets.

PROC DATASETS and the ACCESS window allow you to create an index and define its attributes and type (regular or composite). The IML procedure allows index creation, but only for regular indexes, and the procedure assigns default attributes. PROC SQL also allows for both types of indexes to be created but assumes the index to be unique. To use PROC DATASETS to create an index use the following syntax:

```
PROC DATASETS LIBRARY = CORPDATA;
   MODIFY EMPLOYEE;
   INDEX CREATE LASTNAME;
   INDEX CREATE DEPT / NOMISS;
   INDEX CREATE SSN  / UNIQUE;
   INDEX CREATE NAMEDEPT =( DEPT LASTNAME );
```

```
   INDEX DELETE MANAGER;
RUN;
```

Options available for the INDEX CREATE statement are UNIQUE and NOMISS. The NOMISS option does not permit the index file to contain a missing value. The UNIQUE option allows only one record for each key. There are also several methods of determining the index structure contained in a SAS data set. In display manager, use of the DIR, VAR, or ACCESS windows will provide information about the index if one exists. Detailed information on the indexing structure is also available from PROC DATASETS using the CONTENTS statement or PROC CONTENTS.

```
PROC DATASETS LIBRARY = CORPDATA;
   CONTENTS DATA = EMPLOYEE;
RUN;
```

The following partial output is generated from the above procedure:

```
                    DATASETS PROCEDURE

     -----Alphabetic List of Indexes and Attributes-----

                  Nomiss
#    Index        Option      Var1      Var2
-----------------------------------------------
1    DEPT         YES
2    LASTNAME
3    NAMEDEPT                 DEPT      EMPNAME
```

Once an index or indexes have been created, the SAS System determines when to use them. The SAS System will always use the index for BY processing. For WHERE processing the SAS System will select to use the index if optimization of resources can be attained by use of the index. In other words, if using the index is cheaper than using another indexing strategy or a sequential pass of the entire data set, the SAS System will choose to utilize the index. The use of the SAS system option MSGLEVEL=I can be used to obtain a message in the log as to the use of the index. The following tradeoffs are associated with the use of an index:

- extra CPU cycles and I/O operations to create and maintain the index
- extra memory for page buffers
- extra disk space to store the index data structure
- extra resources to rebuild indexes data for a new version of a data set with indexes
- any manipulation of the SAS data set outside the SAS System must consider the auxiliary index file
- perhaps a less efficient access method being chosen
- if an operation expects to include missing values the index will not be used if the index is defined with the NOMISS option.

Indexes are preserved if a copy of the data set is made using the COPY procedure or PROC DATASETS. The index is not preserved if the copy is made using a DATA step. The following are suggested guidelines for using indexes:

- Keep the number of indexes to a minimum to reduce disk storage and update costs.
- The WHERE cost estimation is most accurate if the key variable's values are uniformly distributed and the minimum and maximum values are consistent with the rest of the data.
- Do not create an index unless the data set size is at least three page buffers (PROC CONTENTS provides this information).
- Do not use the NOMISS attribute if you expect the index to be used to optimize WHERE processing that will select missing values.
- An index works best when it is used to retrieve a relatively small number of records (<50%).

Data File Compression

Release 6.06 introduces the capability to compress SAS data files. In an uncompressed data file all records have the same length. The numeric values are stored as floating point and the character values are stored in fixed length fields, padded with blanks if the value does not fill the field completely. In a compressed data file, each record may have a different length. In compressed data files, the entire record is treated as a single string of bytes with the variable types and boundaries being ignored. Consecutive repeating characters are collapsed into fewer bytes. There are two techniques for compressing files: the COMPRESS= data set option and the COMPRESS SAS system option. The SAS system option causes all subsequent files to be compressed. The CONTENTS of a data file will display the compressed attribute as either YES or NO. To compress the employee master file we code

```
DATA CORPDATA.EMPLOYEE (COMPRESS=YES);
   SET CORPDATA.EMPLOYEE;
RUN;
```

The SAS System can identify whether a data file is compressed or not and handles the data file accordingly. At execution time each record is decompressed for processing. The certain tradeoffs inherent with compressed data files are illustrated here:

UNCOMPRESSED	COMPRESSED
requires more mass storage	requires less mass storage
deleted record space is never reused	deleted record space can be reused
observations are addressable by number	observations are not addressable by number (no SET with a POINT=)
requires less CPU time to prepare records for I/O	requires more CPU time to prepare records for I/O
an updated record will fit in its original location	and updated record may not fit in its original location

CONCLUSION

The logical problem everyone now faces is how to implement these new features to achieve maximum efficiency. There are no hard and fast rules that apply to all situations. The efficiency that may be gained is strictly dependent on all of the following factors:

- operating system
- data file size
- specific application
- priorities of the site.

The general rule of thumb is that for your specific applications you will need to experiment and test various combinations of features to determine exactly what works best to meet your specific requirements.

SAS Institute has several resources available to assist you with any conversion issues you may have. There is new documentation available from the Publications Division. There will continue to be tele-

phone assistance available from the Technical Support Division. The Education Division has revised several of the training courses and added a new two-day course, Making the Transition to Release 6.06 of the SAS System. The recently formed SAS Consulting™ Services group is also available to assist in meeting your company's specific needs.

REFERENCES

SAS Institute Inc. (1990), *Making the Transition to Release 6.06 of the SAS System Course Notes*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1989), *SAS Guide to the SQL Procedure: Usage and Reference, Version 6, First Edition*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1989), *SAS Language, Release 6.06, Preliminary Documentation*, Cary, NC: SAS Institute Inc.

SAS institute Inc. (1989), *SAS Language and Procedures: Usage, Version 6, First Edition*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1987), *SAS Views: SAS Processing*, Cary, NC: SAS Institute Inc.