

Writing a Data Parser Using the SAS® System

Andrew T. Kuligowski, Nielsen Media Research

ABSTRACT / INTRODUCTION

Most SAS users have been assigned a task in which they need to read in data from an external source in order to process it with the tools available under the SAS System. Usually, the format for the data is documented and accompanies the dataset or file to be processed. On occasion, however, a challenge arises. The data exists on electronic means, but it is intermingled with unnecessary clutter - for example, it is contained in an electronic copy of a paper report. This paper will discuss the principles of using the SAS System to parse a file to extract useful data from a normally unusable source. This will be accomplished by citing examples of unusual data sources and the SAS Code used to parse it.

PARSERS - A PRIMER

The word *parser* will normally cause a computer-minded individual to think of a compiler or interpreter. Both must include a parser, which determines the syntactic structure of a string of characters coded in a high-level language. However, while correct, this is too specific a definition for our purposes. Let us use a more generic (read: non-computer specific) definition of *parse* as *the analysis of a string of characters and subsequent breakdown into a group of components*.

To illustrate this definition, let us cite an example which will be familiar to many SAS users. The Copyright statement which appears at the beginning of a SAS Log contains a character string which is unlikely to be used elsewhere in a routine: "Cary". When using an on-line editor to browse a listing which contains a SAS Log, searching for the word "Cary" should bring the user to the start of the SAS Log. [See Figure 1.] This basic

example shows how a unique character string can be used to isolate a specific section of a text file for further processing.

Please note that the examples used in this paper were taken from the MVS environment. The concepts illustrated are independent of platform, and can be applied to any platform on which the SAS System currently resides.

PARSER EXAMPLE # 1 - An IDCAMS Listing

As our first example of a data parser, we will remove the MVS dataset names from the output of an IDCAMS statement. (By definition, IDCAMS is a program interface to the Access Method Services facility for VSAM files under the MVS environment. However, the facility can also be used for non-VSAM files.) The following command was issued from an IDCAMS MVS step:

```
LISTCAT LEVEL (ACCOUNT.LOOKUP.FILE)
```

The SYSPRINT was captured in a sequential file [See Figure 2.], and passed to a SAS Data Parser.

The key to any successful parser is the ability to identify and utilize patterns in the material being parsed. In this example, we can easily identify one such pattern - every line of our listing containing a valid dataset name precedes that name with the word "NONVSAM" and a series of dashes. Therefore, our parser must look for the presence of the character string "NONVSAM" in position 2 of each line of output - position 1 being reserved for carriage control characters. If a line passes this test, we read in the next two "words", which will be a series of dashes and an MVS dataset name. (All lines failing this test are ignored.)

```
BROWSE ----- USERID.SASRUN.LISTING ----- CHARS ' Cary' FOUND
COMMAND ==>                                SCROLL ==> CSR

.NOTE: Copyright(c) 1989 by SAS Institute Inc., Cary, NC USA.
.NOTE: SAS (r) Proprietary Software Release 6.08 TS407
.      Licensed to A COMPANY, Site #####.
.
.NOTE: Running on IBM Model 3090 Serial Number #####.
.
.      Welcome to the SAS Information Delivery System.
.
```

Figure 1
Searching for the word "Cary" at the top of a SASLOG

```

1IDCAMS  SYSTEM SERVICES
0
  LISTCAT  LEVEL (ACCOUNT.LOOKUP.FILE)
ONONVSAM ----- ACCOUNT.LOOKUP.FILE
  IN-CAT --- CATALOG.APPL1
ONONVSAM ----- ACCOUNT.LOOKUP.FILE.D941227
  IN-CAT --- CATALOG.APPL1
ONONVSAM ----- ACCOUNT.LOOKUP.FILE.D941228
  IN-CAT --- CATALOG.APPL1
ONONVSAM ----- ACCOUNT.LOOKUP.FILE.D941229
  IN-CAT --- CATALOG.APPL1
                                     [Text removed for brevity]
ONONVSAM ----- ACCOUNT.LOOKUP.FILE.D950211
  IN-CAT --- CATALOG.APPL1
ONONVSAM ----- ACCOUNT.LOOKUP.FILE.D950212
  IN-CAT --- CATALOG.APPL1
1IDCAMS  SYSTEM SERVICES
0      THE NUMBER OF ENTRIES PROCESSED WAS:
                                     [Text removed for brevity]
      TOTAL -----72
0      THE NUMBER OF PROTECTED ENTRIES SUPPRESSED WAS 0
0IDC0001I FUNCTION COMPLETED, HIGHEST CONDITION CODE WAS 0
0
0IDC0002I IDCAMS PROCESSING COMPLETE. MAXIMUM CONDITION CODE WAS 0

```

Figure 2
SYSPRINT returned from IDCAMS LISTCAT step

```

DATA ALL_REF(KEEP=REFNAME);
  LENGTH DASHES $ 8.;
  REFNAME $ 44.;
  INFILE REFLIST;
  INPUT @ 2 NONVSAM $CHAR7. 8;
  IF NONVSAM ^= 'NONVSAM' THEN RETURN;
  /* Else Do (Implied) */
  INPUT DASHES
  REFNAME;
  /* Various error tracking and reporting tasks */
  /* occur here. They have been removed from */
  /* this version of the routine for brevity. */
  OUTPUT;
RUN;

```

Figure 3
SAS Routine to parse MVS dataset names from IDCAMS LISTCAT output

Let us examine the SAS code that performs this task. [See Figure 3.] The "line-hold specifier", which is the trailing "@" at the end of the input statement, is the key feature for this task. By default the INPUT statement will process one line of data. The trailing "@" will cause the INPUT statement to hold a line of data, rather than moving on to the next line. This allows the routine to check for the presence or absence of a given character string. If the "NONVSAM" string is present, then the next INPUT statement reads additional information from the same line of data. However, if the "NONVSAM" string is

not present, a subsequent INPUT statement, without variables or the trailing "@", causes the INPUT statement to release the held line and move to the next line of data. (Please note that this INPUT statement is included in the sample code to illustrate this phenomena. A line of data which is held by a trailing "@" is released when the system returns to the top of the DATA Step to begin the next iteration. The INPUT statement must end with a "double trailing @", or "@@", to circumvent this automatic release.)

PARSER EXAMPLE # 2 - A SASLOG Containing a PROC PDS

We will now move to a more complex example, one that must parse more than one piece of information from each line of usable data. Our next parser will remove the member names from an MVS Partitioned Dataset (PDS) from a SASLOG generated using PROC PDS.

In order to execute the parser, we need input. This is obtained from the following one line SAS PROC:

```
PROC PDS DDNAME=SOMEFDS; RUN;
```

The SASLOG is captured in a sequential file [See Figure 4], and passed to the next routine, which will parse the data. Please note that this PROC *must* be executed in a separate invocation of the SAS System, in order to close the dataset containing the SASLOG.

The SAS code used to parse the PDS member names differs from the earlier example. [See Figure 5.] The variable PARSEKEY was aptly named, as it is the main entity of the routine. At the start of the routine, PARSEKEY is set to "MEMBERS". A close examination of the SASLOG shows that this word *only* occurs in the line immediately preceding the list of PDS members. Therefore, the routine is set up to look for this character string; all lines preceding the one that contains that word are discarded without further processing. (Once the routine has located this string, the word loses its significance. Therefore, should the PDS contain a member actually called "MEMBERS", the routine will handle it properly.)

Once the word "MEMBERS" is located, the routine performs two tasks. First, a field called PARSEFLG is set to "Y". This will cause the parser to process all subsequent lines in the SASLOG. In addition, the value of PARSEKEY is redefined to "TRACKS USED". Again, an examination of the SASLOG shows that this string only occurs once - in the first non-blank line immediately following the list of PDS members. The purpose of this string has also been subtly altered; the field that was used to inform the routine to *begin* parsing will now be used to advise the program to *stop* parsing.

At this point, the routine performs the bulk of its task. Each line that is read in - assuming it does not contain the now-redefined PARSEKEY - will contain a number of valid PDS member names, separated by one or more blank characters. The routine uses two SAS functions to obtain this data. The INDEX function is used to determine the position of the first blank character in the input line. The SUBSTR function is then employed to strip the member names from each line of data. Two other SAS constructs complete this section of the routine. The OUTPUT statement is used to create an

observation in the SAS dataset containing the member name. A DO WHILE statement is used to ensure each line of the SASLOG is fully processed - as long as the first blank character in the input line is not in position 1, the routine will continue to parse for PDS member names.

WARNING - Do You Really Need to Write a Parser?

This paper would be incomplete without a warning: There are often better alternatives to a data parser. In fact, the two examples cited in this paper could have been written using an alternate method! For starters, the IDCAMS Listing from Example # 1 was originally written as part of a process that could verify the existence of a dataset. However, the DSNEXST command, which was added to Version 6 of the SAS System, can be used for the same purpose using only a few SAS statements.

Similarly, the PROC PDS parser cited in Example # 2 could have been written by opening the PDS with a SAS FILE statement as follows:

```
FILE pdsref LRECL=256 BLKSIZE=256;
```

This will cause SAS to process the PDS's directory, rather than any of the individual members.

Finally, do not eliminate manual intervention from consideration. In many cases, it would be quicker to have someone manually type the data into a sequential dataset than it would be to write and test a parsing routine to automate the process. This is especially true for "one-time-only" requests; if the request is for an ongoing process, a parser is more likely to be cost justified.

CONCLUSION

A data parser can be an effective tool to extract useful data from a normally unusable source. The prospective author of a data parser using the SAS System should become well acquainted with the syntax and options for the INFILE and INPUT statements. They should also be prepared to make judicious use of the DO UNTIL and DO WHILE statements. Additionally, before commencing on a potentially complex coding project, they should be certain that a simpler solution might not be available. However, when necessary and done properly, a data parser can be a blessing, providing availability to data which might have been thought inaccessible.

```

11                               The SAS System                16:16 Friday, January 6, 1995

NOTE: Copyright(c) 1989 by SAS Institute Inc., Cary, NC USA.
NOTE: SAS (r) Proprietary Software Release 6.08 TS407
      Licensed to A COMPANY, Site xxxxxxxxxxxx.

NOTE: Running on IBM Model xxxx Serial Number xxxxxx,
      IBM Model xxxx Serial Number xxxxxx,
      IBM Model xxxx Serial Number xxxxxx,
      IBM Model xxxx Serial Number xxxxxx,
      IBM Model xxxx Serial Number xxxxxx.

Welcome to the SAS Information Delivery System.

This message is displayed for users in the SAS log when the NEWS
option is specified. You can replace this message with your own by
editing the NEWS file.

NOTE: The SASUSER library was not specified. SASUSER library will now be
      the same as the WORK library.
NOTE: All data sets and catalogs in the SASUSER library will be deleted at
      the end of the session. Use the NOWORKTERM option to prevent their
      deletion.

NOTE: SAS system options specified are:
      MACRO SORT=4 LOG=SASLOG

NOTE: The initialization phase used 0.31 CPU seconds and 3402K.
1      PROC PDS DDNAME=SOMEPPDS; RUN;

SAS PROC PDS VERSION 6.08    10/21/93

DSNAME=ACCOUNT.OTHER.STUFF.SOMEPPDS VOL=SER=xxxxxxxx

Members (aliases)

MEMBER1A MEMBER1B MEMBER1C MEMBER1D MEMBER1E MEMBER1F MEMBER2A MEMBER2B
MEMBER3A MEMBER4A MEMBER5A MEMBER5B MEMBER5C MEMBER5D MEMBER5F MEMBER5G
MEMBER5H MEMBER5I MEMBER5J MEMBER5K MEMBER6A MEMBER6B MEMBER6C MEMBER6D
MEMBER6E MEMBER6F MEMBER7A MEMBER7B MEMBER7C MEMBER7D MEMBER7E MEMBER7F
MEMBER8A MEMBER8B MEMBER8C MEMBER8D MEMBER8E MEMBER8F MEMBER8G MEMBER8H
MEMBER9A MEMBER9B

Tracks Used      644.9
      Unused      7.1
      Total      652.0
Extents          3

Directory Blks   30
Blocks Used      27

NOTE: The PROCEDURE PDS used 0.11 CPU seconds and 3440K.

NOTE: The SAS session used 0.43 CPU seconds and 3440K.
NOTE: SAS Institute Inc., SAS Campus Drive, Cary, NC USA 27513-2414

```

Figure 4
SASLOG generated from PROC PDS

```

DATA PDSMEM(KEEP=MEMNAME);
  LENGTH PARSEKEY $ 16. PARSEFLG $ 1. INLINE $ 133.
  MEMNAME $ 8. ;
  RETAIN PARSEKEY 'MEMBERS' PARSEFLG 'N';
  /* Input a line from the SASLOG, convert to UPPER CASE. */
  INFILE OLDLOG END=ENDLOG;
  INPUT @ 1 INLINE $ 133. ;
  INLINE = UPCASE( INLINE );
  /* Watch for the "Parse Key", which will indicate that we */
  /* are either to begin parsing or stop parsing. */
  IF INDEX( INLINE, TRIM(PARSEKEY) ) = 0 THEN DO;
    IF PARSEFLG = 'N' THEN DO;
      /* When the program starts, the "parse key" is set to a */
      /* string that will indicate the end of the extraneous */
      /* header information i.e. the start of the "valuable" */
      /* data. Ignore all input prior to that point. */
      DELETE;
    END;
    ELSE DO;
      /* We have a line with PDS member names. Loop through the */
      /* line of data, writing each PDS member name to a separate */
      /* observation. */
      B_INDEX = INDEX( INLINE, ' ' );
      DO WHILE( B_INDEX > 1 );
        MEMNAME = SUBSTR( INLINE, 1, B_INDEX );
        INLINE = LEFT( SUBSTR( INLINE, B_INDEX + 1 ) );
        B_INDEX = INDEX( INLINE, ' ' );
        OUTPUT;
      END;
    END;
  ELSE DO;
    /* We have found the "Parse Key". If we have not yet begun */
    /* parsing, this is the sign that we should now start. If */
    /* we have been parsing, this is the indication that we have */
    /* reached the end of our data - only extraneous trailer */
    /* information remains - and we should stop the routine. */
    IF PARSEFLG = 'N' THEN DO;
      PARSEFLG = 'Y';
      PARSEKEY = 'TRACKS USED';
      INPUT ;
    END;
    ELSE STOP;
  END;
RUN;

```

Figure 5
SAS Routine to parse PDS member names from PROC PDS output

REFERENCES / FOR FURTHER INFORMATION

Augustine, Michael (1995). "Processing Legacy Reports". Presented at the 13 June, 1995 meeting of the New York Area SAS Users Group (NYASUG).

Beach, John (1994). "Undocumented Features to Test for Existence of Data Sets". *SUGI 19 Proceedings*, p 144. Cary, NC: SAS Institute, Inc.

Brinsfield, Eric (1988). "Interacting With Your Host System From the SAS System Under OS". *SUGI 13 Proceedings*, p. 522. Cary, NC: SAS Institute, Inc.

Calvert, William S. and Rao, Malla R. (1992). "A SAS Code Formatter". *SUGI 17 Proceedings*, p 158. Cary, NC: SAS Institute, Inc.

Itano, Glenn. (1994). "re: Extracting PDS Member dates on MVS". SAS-L Posting of 23 May 1994.

Kuligowski, Andrew T. (1994). "IDCAMS to SAS - The Parser Two-Step". *SESUG '94 Proceedings*, p. 82. Cary, NC: SAS Institute, Inc.

SAS Institute, Inc. (1990). *SAS Language: Reference, Version 6, First Edition*. Cary, NC: SAS Institute, Inc.

Sinclair, Ian R. (1991). *The Harper-Collins Dictionary of Computer Terms*. New York: Harper-Collins.

Webster's New World Dictionary of Computer Terms. (1988). New York: Webster's New World.

SAS is a registered trademark of SAS Institute, Inc. All rights reserved. * indicates USA registration. IDCAMS and VSAM are registered trademarks of International Business Machines Corporation.

ACKNOWLEDGMENTS

Thanks to Luan Ardinger, Michael Augustine, William Calvert, F. Joseph Kelley, Andy Parks, David Riba, Joe Ruberte, and Helen-Jean Talbott, for their thoughts on the contents of this paper and corresponding poster and/or for agreeing to proofread its myriad drafts. Thanks also go out to Karen Rehg, Young Ta and Ken Wood, for their contributions to this paper.