CONVERTING SPSS-X* SYSTEM FILES TO SAS DATASETS

Grant Blank, Independent Consultant and Andrew A. Norton, National Opinion Research Center

I. ABSTRACT

An extended version of SPSS*, named SPSS-X, will replace the existing SPSS release in early 1983. The new system files are substantially different from older SPSS files and they cannot be read by PROC CONVERT. Thus, conversion of the new files will require writing the file in raw data form, constructing SAS control cards, and reading the file into SAS with all the associated problems and cost of debugging, typing variable names, input formats and labels for hundreds or even thousands of variables. To avoid this arduous process, we have written a program, named SPSSXSAS, which will read the dictionary in the SPSS-X file and use that information to create, automatically, a SAS dataset. SPSSXSAS will convert variable names, labels, print formats and missing values; and, in a valuable extension of the capabilities of PROC CONVERT, it will convert value labels and documents.

II. CONTEXT OF THE PAPER

SPSS has achieved wide use as a statistical package, but it has always been hampered by its primitive data management capabilities. An extended version of SPSS (called SPSS-X), containing capabilities similar to SAS's SET and MERGE statements as well as other enhancements, is scheduled for release in February 1983. When SPSS-X is released, all older releases of SPSS will be superseded and no longer available from SPSS, Inc. The system files used by SPSS-X are substantially different from older SPSS system files and they cannot be read by SAS's PROC CONVERT. This means that programmers will find it difficult and costly to use the same data on both systems.

The program we have written, SPSSXSAS, will convert SPSS-X files to SAS datasets. The program design is based on the philosophy of attempting to preserve as much information as possible through the conversion process. Thus, in addition to converting variable names, labels, print formats, missing values and data, it extends the capabilities of PROC CONVERT to convert value labels and documents.

SPSS-X is designed to run on many different types of computers. This has a number of implications concerning the way in which system files are organized and we will point to several in the course of this paper.

This paper is addressed to two audiences. One audience is SAS users who want to use data stored in an SPSS-X system file which was created at their installation or at another installation with the same hardware and operating system. A second audience consists of SAS programmers who are interested in learning new programming techniques. In the course of writing SPSSXSAS, we have developed a number of techniques that we think are quite useful but not commonly used.

III. SPSS-X SYSTEM FILES

Introduction

To understand the development of the program some understanding of the structure of SPSS-X system files is required. The information that follows is based on our understanding of an SPSS-X system file. As far as we know, there is no publicly available documentation on internal SPSS-X system file organization.[1]

The important thing to keep in mind during the following discussion is that the internal organization of an SPSS-X system file does not correspond to that of any standard IBM file organization.[2] To an IBM operating system, a system file has the following characteristics:

```
   RECFM = FB
   LRECL = 1024
 BLKSIZE = default of 3072
```

Like SAS data sets and old SPSS files, each of the new system files begins with a dictionary containing variable names, labels, formats and other information describing the file. Following the dictionary, are the data records. Although the file is defined as record format FB, the internal structure of both the dictionary and the data does not correspond to the logical records.

Dictionary Segments

Essentially, the internal structure of the dictionary of a system file is built around a subunit which we will refer to as a "segment."

The dictionary contains at least six different types of segments, where each type contains different information. Segment types in SPSS-X system files roughly correspond to the record types in a SAS dataset. The segments in the dictionary may be shorter or longer than a logical record, depending on the information that they contain. Each segment begins with a one-word (i.e., four-byte) identifier that tells what type of information it contains. The segments and the types of information that they contain are listed in Table 1.

TABLE 1

Segment Types*

Identifier	Information
FL2	File Definition
2	Variable Definition
3	Value Labels
4	Indices of Variables Associated with above Value Labels
6	Documents
999	Start of Data

*We have never seen segments labeled type 1 or type 5.

It is not relevant to the purpose of this paper to describe each segment type in detail, but several characteristics are noteworthy.[3] First, almost all information is stored as single words (i.e., four bytes), even when it could be stored as a single bit. This is apparently one of the results of SPSS's determination to make SPSS-X compatible across many types of computers. Second, individual segments may span one or more logical records. Third, except for the File Definition segment (segment type FL2), segments may vary in length, depending on the information that they contain. In fact, the Value Label (segment type 3), Indices for Value Labels (type 4) and File Documentation (type 6) segments may be longer than a single logical record; potentially longer than several logical records. Finally, segments 2, 3, and 4 generally appear multiple times. Each variable and value label is defined in a separate segment.

Data Cases

The data portion of an SPSS-X system file begins following segment type 999. Data are stored in fixed length sections which we shall refer to as "cases". Cases span logical records and may be longer than one or more logical records.[4] This structure has the major advantage that it allows SPSS-X files to store a virtually unlimited number of variables. In contrast, SAS datasets are limited by the number of variables that can be stored in the maximum length allowed for a single logical record--about 4000 double-precision variables.

SPSS-X always stores numeric data in floating point double precision (SAS format RB8.), except in compressed files. Character data is stored in EBCDIC character code. The length of all character variables is always the nearest multiple of eight larger than or equal to the actual length of the variable. For example, even if a variable is only one byte long, it will be stored in eight bytes. This appears to be another effort to remain as compatible as possible across many different types of computers.

These and other less central issues created problems for us as we tried to read these files without access to documentation of the internal structure. Our solutions contain a number of techniques which are not in common use among SAS programmers, even though they are potentially quite useful. We present them below in the context of the development of the program.

IV. DEVELOPMENT OF THE PROGRAM

Overall Organization of the Program

Our overall conception of SPSSXSAS was to write a single DATA step which would read the dictionary of the SPSS-X file and output SAS statements to a temporary file. %INCLUDE statements would be used to incorporate these SAS statements into a second DATA step which would read the data section of the SPSS-X file as INPUT data, converting it to a SAS dataset. The core of our program is the first DATA step which generates the SAS code. The division of the dictionary into six segment types allowed us to organize this DATA step in the form of one subroutine for each segment type and a supervisor. The primary function of the supervisor is to read the field containing the segment type number and pass execution to the appropriate segment type subroutine through a LINK statement. Each of the subroutines is designed to read its segment and translate the information into SAS statements which are output to a temporary file, then return to the supervisor.

Buffered Input

A major problem stems from the fact that dictionary segments and data cases may begin or end at virtually any word in a logical record and often span logical records. When they span logical records, dictionary segments and data records are broken at byte boundaries, but not necessarily at word boundaries. Since most of the information in an SPSS-X file is stored in double-precision (i.e., two four-byte words--eight bytes), an individual variable may be split between different logical records. For character variables, the parts need to be read separately, then concatenated together. For numeric variables, a character buffer is required so that the parts may be concatenated together and then converted to numeric form.

This suggests an approach. We could read split variables into a character buffer, then use substrings and the INPUT function to convert them to numeric form. One way to implement this approach would be to input variables directly except when they cross record boundaries, in which case a buffer would be used.

This approach would require writing conditional INPUT statements able to read every possible pattern of variables that may be encountered in a file, using some sort of special processing to read across record boundaries. This is feasible only if the range of possible patterns is limited. The number of possible patterns is complicated by two characteristics of SPSS-X files. First, since segments and cases may begin or end anywhere and often span logical records, there is no simple way to determine where they will begin or when they will be split. Second, the format of the dictionary segments is far from fixed field. Fields inside of a segment must frequently be evaluated in order to determine what format follows.

As we discussed this approach, we quickly realized that even though multiple INPUT statements are very flexible, they could not feasibly handle skipping to a new record in the middle of any variable in any segment or case. The potential number of patterns of variables and record skips is virtually infinite. This problem was particularly acute in the data portion of the file because the data are read by the second DATA step from code generated by the first DATA step. The rules that the first DATA step would have to follow in writing the INPUT statements quickly became unmanageably complex.

A far simpler approach is to buffer everything. This is the approach that we chose. All input is copied into a buffer, and then variables are loaded by substringing from the buffer and converting if necessary. These conversions are generally inexpensive. Character variables require a simple substring operation. SPSS-X numeric variables are stored in RB8. double precision real binary format. Since this is also how SAS stores numeric data, no conversion is required, just redefinition as numeric variables.

In SPSSXSAS, the buffer consists of a character variable of length 200 which is loaded in a separate subroutine. Before this subroutine is LINKed, a variable is set to the number of characters to be loaded into the buffer. Whenever the number of characters requested exceeds the characters remaining on a logical record, the buffer is loaded to the end of the current logical record and the remaining characters are added from the next record. This solves the problem of locating a field on the file. Regardless of the location of a field on a record (or on two records if the field spans records), the field is always in a fixed position in the buffer. A single substring operation will always be able to read it. This approach also solves the problem of variable input formats. A variable length section of data may be obtained simply by setting the number of characters requested from the buffer equal to the value of a variable or expression.

The use of a separate subroutine for input gives us important advantages. Since data are INPUT at only one location in the program, any problems with incorrect or invalid input are easy to locate. The special processing required to skip to the next record needed to be written and debugged only once. The code in the subroutine that fills the buffer is short, readable and easy to modify. Furthermore, in the extreme case when SPSSXSAS is converting a file containing 4000 variables, we must limit the number of lines of code in the second DATA step to an average of seven or less per variable. SAS is restricted to 32,000 command lines in a single job step and the first DATA step uses nearly 1,000 lines. A separate subroutine to handle INPUT helps SPSSXSAS save lines.

Buffer INPUT Efficiency

Once we had decided to input all data through a single buffer, the efficiency with which we could fill that buffer became an important issue. Our solution to the problem of efficiently filling the buffer responds to characteristics of SPSS-X files that, by now, are quite familiar to the reader: that segments and cases can span logical records and that the breaks across logical records will occur only at byte boundaries.

At this stage, we were working with an earlier version of SPSS-X in which the breaks across records occurred only at word boundaries. Our original solution was to INPUT one word (four bytes) at a time. This avoided the problem of an INPUT statement that would cross record boundaries, but it meant that putting 200 characters into the buffer would require that the INPUT statement be executed fifty times. It worked, but all those INPUT statements were time consuming.

Then we discovered the $VARYING informat and this became the key that enabled us to efficiently fill the buffer. This informat allows us to INPUT a varying length character string, with its length controlled by the value of a numeric variable. Except when the number of characters requested exceeds the number of characters remaining on a record, this variable equals the full length requested. The only tricky part occurs when we must load the buffer over two different logical records. In this situation, we calculate the length remaining to the end of the current record, load the buffer with the first portion of the data, and then use another INPUT statement to load a secondary buffer with the remainder. A pseudostring (target substring) is then used to move the data from the secondary buffer into the main buffer.

The $VARYING informat allows us to reduce the number of INPUT statements from fifty to one (or, at most, two--when the INPUT crosses record boundaries). We are able to save significant amounts of time, particularly when we are reading the data records.

Reading Data Cases

The data cases are read using the same subroutine to fill a buffer. These cases differ from the dictionary segments in that both the length and the layout of a case are fixed, and known from the information supplied by the dictionary. Except for very small files, the cases are more than 200 bytes long and the program must LINK to fill the buffer as many times as are necessary to read a single case.

Unlike the dictionary segments which reuse the same 200-character variable as a buffer, the data cases are read from multiple buffers. This lets us take advantage of the dictionary information to improve efficiency with which the data are INPUT. The buffers are arranged so that an entire case is INPUT into them and then individual variable values are extracted using substring functions.

The buffers are defined as we read segment type 2 in the dictionary. In segment 2, SPSSXSAS writes two sections of SAS code concurrently. One is the statements required to load the buffers; the other is the code to read the variables from the buffers.

To be efficient, we would like to fill each buffer with as close to 200 characters as possible. This creates a problem. As each variable definition segment (type 2) is being read, SPSSXSAS writes out the code for that variable. But we do not know how many variables will fit into each buffer until we read a variable with a length that would extend beyond the buffer. If the code to load the buffer was written at this point, it would follow the variables that are supposed to use it for input. Obviously, this wouldn't work because the buffer must be filled before variables can read it.

We solved this problem by writing the variable definitions to one temporary file and the statements to load the buffers to a second. Use of %INCLUDE statements allows us to reverse the order of the statements in the second DATA step.

V. BENCHMARKS

SPSSXSAS has been implemented and tested on the University of Chicago's Amdahl 470 V/7 under OS/SVS and SAS 79.6. We ran benchmarks using 10, 100 and 1000 variables and 10, 100 and 1000 cases. In each test one out of every ten variables was a character variable, the rest numeric. The test files included no labels or documents and no data were missing. Table 2 gives the time and region used by each of the DATA steps in SPSSXSAS.

This table compares the time and region required by SAS and SPSS-X to read raw data and convert it to a dataset or system file with that required by SPSSXSAS when it converts an SPSS-X system file. All three programs were working on identical data as described above.

Table 3 contains some surprising results. First, a comparison of the amount of time required by SAS with that required by SPSS-X demonstrates that SAS can create a small dataset faster than SPSS-X. But large SAS datasets require much more time to create than do equally large SPSS-X system files. In the extreme case of 1000 variables and 1000 cases, SAS required over twice as much time. Second, the effects of compiling all those SPSSXSAS statements show up here too. The SAS program that read 1000 variables was the same length as the SAS program that read 10 variables--eight lines. By comparison, the SPSSXSAS program that reads 1000 variables is over 5000 lines longer than the program that reads ten variables. For 1000 variables, the difference between the time that SAS requires to read ten cases and 1000 cases is 19.6 seconds. For SPSSXSAS, the difference is 29.8 seconds.

TABLE 2

Time and Region Benchmarks for SPSSXSAS

Number of Variables

		10			100			1000		
		Cases			Cases			Cases		
		10	100	1000	10	100	1000	10	100	1000
Read Dict. & Output SAS Code	Time	.6	.6	.6	.7	.7	.8	1.9	1.9	1.9
	Region	332	332	332	336	336	336	384	384	384
Read Cases	Time	.2	.3	.6	1.1	1.3	3.9	9.1	11.8	39.0
	Region	208	208	208	216	216	216	484	484	484

Units: Time is expressed in seconds of CPU, region is in Kilobytes.

Several important insights are apparent in Table 2. First, notice the economies of scale that SPSSXSAS achieves in reading the dictionary. Processing 1000 variables requires only about three times more time than does processing 10 variables. Second, similar economies of scale are evident when SPSSXSAS processes cases. Regardless of the number of variables, the time required to process 1000 cases is only about three times as much as that required for 10 cases. Finally, processing many variables requires a significant overhead. When converting 10 cases, 1000 variables requires almost nine seconds more than does ten variables. The major element here is the time needed to compile the SAS statements that read 1000 variables. There are over 5000 lines of code in the DATA step that reads the cases for 1000 variables.

Table 3 compares SPSSXSAS with SAS and SPSS-X.

But the additional overhead that SPSSXSAS pays to compile the 1000 variables is constant. Thus, while SPSSXSAS requires eleven times more time than SAS to process ten cases, it only requires twice as much time to process 1000 cases. Finally, looking across Table 3 we see that SPSSXSAS requires two to five times more time than SAS to create the same dataset. Because SPSS-X runs faster than SAS when creating large files, SPSSXSAS needs four to seven more times more time than SPSS-X.

VI. CONCLUSION

The value of our program can be best illustrated by a realistic comparison. Consider the situation where a programmer is preparing files for two sets of users--some who want to use SPSS-X and others using SAS. The programmer

TABLE 3

Comparative Benchmarks

Number of Variables

		10			100			1000		
		Cases			Cases			Cases		
		10	100	1000	10	100	1000	10	100	1000
SPSSXSAS										
	Time	1.3	1.3	1.6	2.2	2.6	5.1	11.5	14.1	41.3
	Region	364	364	364	364	364	364	484	484	484
SAS										
	Time	.1	.1	.3	.1	.3	2.1	1.0	2.8	20.6
	Region	188	188	188	204	204	204	324	324	324
SPSS-X										
	Time	.2	.2	.2	.4	.3	.4	1.2	2.2	9.4
	Region	448	448	448	448	448	448	512	512	512

The Time and Region data are the total time and the largest region required during the entire job. Time is in seconds, Region in kilobytes.

will want to create two copies of each file; one in the form of an SPSS-X system file and the other a SAS dataset. Here the appropriate comparison is the sum of the CPU time that would be used by SAS and SPSS-X to create the two files versus the combined time used by SPSS-X and SPSSXSAS. For example, to create identical SAS and SPSS-X datasets from raw data containing 1000 cases and 1000 variables would cost 30 seconds using SPSS-X and SAS. Using SPSS-X and SPSSXSAS would require 50.7 seconds. The difference between the two approaches is 20.7 seconds.

This comparison illustrates the value of SPSSXSAS. The additional 20 seconds of CPU time is dramatically less costly than is writing and debugging the hundreds or thousands of SAS statements required to INPUT a file containing 1000 variables.

VII. WHERE TO CONTACT THE AUTHORS[5]

Comments or suggestions about SPSSXSAS are welcome and solicited. Copies of the program are available.

Grant Blank
Independent Consultant
Apartment 922
1450 East 55th Place
Chicago, Illinois 60637
(H) 312/947-8194
(W) 312/241-6666

Andrew W. Norton
National Opinion Research Center
6030 South Ellis Avenue
Chicago, Illinois 60637
(H) 312/288-0507
(W) 312/962-1189

VIII. ACKNOWLEDGEMENTS

The development and testing of this program was supported in part by a grant from the Computation Center of the University of Chicago. We would particularly like to thank Harold C. Bloom, Manager of Application Systems, and James Lichtenstein, Programmer/Analyst, for their assistance.

IX. FOOTNOTES

*SPSS and SPSS-X are registered trademarks of SPSS, Inc.

[1]We have obtained all of the information in this paper by creating system files in SPSS-X and then running SAS jobs using the LIST statement to obtain hex printouts of the contents of the file. Our lack of access to documentation from SPSS-X, of course, leaves open the possibility of error, but we have tested our program extensively and it has been able to handle all of the problems that we have given it. We have received no assistance, direct or indirect, from SPSS, Inc. or from SAS Institute.

[2]This, as you may well imagine, made things much more challenging for us as we were reading our hex printouts of the system files.

[3]Detailed information may be found in the program available from the authors.

[4]Note that SPSS-X has the capability to "compress" the data portion of a system file. Our program cannot read compressed files and this description refers only to uncompressed files.

[5]Note: The co-authors' names are listed in alphabetic order.