Paper 177-29

Widefiles, Deepfiles, and the Fine Art of Macro Avoidance

Howard L. Kaplan, Ventana Clinical Research Corporation, Toronto, ON

ABSTRACT

When data from a collection of datasets (widefiles) having similar keys is reorganized as one consolidated dataset having one record per data point, attached to sort keys and variable identifiers, that organization is a deepfile. Associated with the deepfile is a data dictionary that allows many important program operations to be specified by table entries in the dictionary rather than by explicit code. Here, the concepts are illustrated with a few datasets from a pharmaceutical study where information is collected from patients on multiple occasions.

Sorting the deepfile by variable identifier and processing it using by-group logic is an attractive alternative to macros for repeating the same univariate processing over multiple variables. Alternatively, sorting by patient and then occasion facilitates a simple method for listing the entire dataset in the order of events within patients. Deepfiles also facilitate adding new variables, rearranging the order of variables, and exporting data in ASCII formats. These techniques should be of interest to intermediate level programmers working with Base SAS®.

INTRODUCTION

Data from a research study are typically organized into separate datasets, where the datasets share keys for linking records but otherwise have different structures. Each dataset, each record type, represents the different dimensions of a single domain, and each domain has its own dataset. In the pharmaceutical industry, these domains include patient demographics, patient history, vital signs, laboratory tests, and measures of drug concentration. Within each domain, all of the data collected on any occasion share a database record keyed to that occasion.

A dataset structure for a pharmaceutical study might include three datasets such as these. The first dataset identifies which treatment each patient received, the second records vital signs at selected days and times, and the third records the concentrations of a drug and its two principal metabolites at selected days and times (not necessarily the same days and times as the vital signs). In each dataset, any of PATIENT, DAY, and TIME which appear are considered part of the record key:

Dataset DECODING		Dataset VITALS		Dataset DRUGCONC	
PATIENT	Numeric	PATIENT	Numeric	PATIENT	Numeric
		DAY	Numeric	DAY	Numeric
		TIME	Numeric	TIME	Numeric
TREATMENT	Alpha	SYSTOLIC	Numeric	DRUGPARENT	Numeric
		DIASTOLIC	Numeric	DRUGMETAB1	Numeric
		HEARTRATE	Numeric	DRUGMETAB2	Numeric
		BREATHRATE	Numeric		
		SKIN	Numeric (0='Normal', 1='Flushed')		
		COMMENT	Alpha		

Here are several records from each dataset, using that organization:

VIEWTABLE: Work.Decoding

	PATIENT	TREATMENT
1	1	Placebo
2	2	Active
3	3	Placebo

VIEWTABLE: Work.Drugconc

	PATIENT	DAY	TIME	DRUGPARENT	DRUGMETAB1	DRUGMETAB2
6	2	1	14:00	2.61	0.74	0.22
7	2	2	14:00	3.16	0.88	0.30
8	2	3	14:00	3.63	1.01	0.32

VIEWTABLE: Work.Vitals

	PATIENT	DAY	TIME	SYSTOLIC	DIASTOLIC	HEARTRATE	BREATHRATE	BODYTEMP	SKIN	COMMENT
7	1	3	8:00	120	75	60	12	36.6	Normal	
8	1	3	13:00	124	74	66	7	36.8	Flushed	Patient uncomfortable
9	1	3	18:00	126	76	66	7	36.7	Normal	

In the database world, this organization is called "third normal form". The essential features of third normal form are these:

- every record has a unique key (perhaps a multi-part key) within its dataset
- all records within the dataset have the same structure

- no variable's value can be completely predicted by any other variable(s) within the record.

Although the small samples above do not show it, the DAY field is not redundant: a TIME such as 8:00 can occur on both DAY 1 and DAY 2. However, if DAY 1 had only morning data while DAY 2 had only afternoon data, then in terms of information one could eliminate the DAY variable entirely (although there might be other reasons to keep it).

In this paper, we will call that kind of dataset organization "widefiles": in a vertical list of records, both the key and the non-key variables are spread out horizontally across the page. However, an alternate dataset organization captures all of the same information in a form that often makes programming easier. This organization is the "deepfile", and it represents each non-key field of the original widefiles in a separate record:

VIEWTABLE: Work.Deepfile

	PATIENT	DAY	TIME	TREATMENT	DATASET	VARNAME	VARTYPE	VARSORT	NUMERICVAL	ALPHAVAL
16	2	1	8:00	Active	VITALS	SYSTOLIC	NUMERIC	1	119	119
17	2	1	13:00	Active	VITALS	SYSTOLIC	NUMERIC	1	112	112
916	2	1	8:00	Active	VITALS	DIASTOLIC	NUMERIC	2	89	89
917	2	1	13:00	Active	VITALS	DIASTOLIC	NUMERIC	2	83	83
1816	2	1	8:00	Active	VITALS	HEARTRATE	NUMERIC	3	51	51
1817	2	1	13:00	Active	VITALS	HEARTRATE	NUMERIC	3	51	51
2716	2	1	8:00	Active	VITALS	BREATHRATE	NUMERIC	4	10	10
2717	2	1	13:00	Active	VITALS	BREATHRATE	NUMERIC	4	10	10
3616	2	1	8:00	Active	VITALS	BODYTEMP	NUMERIC	5	36.5	36.5
3617	2	1	13:00	Active	VITALS	BODYTEMP	NUMERIC	5	37	37.0
4516	2	1	8:00	Active	VITALS	SKIN	NUMERIC	6	0	Normal
4517	2	1	13:00	Active	VITALS	SKIN	NUMERIC	6	0	Normal
5416	2	1	8:00	Active	VITALS	COMMENT	ALPHA	7	.	
5417	2	1	13:00	Active	VITALS	COMMENT	ALPHA	7	.	
6306	2	1	14:00	Active	DRUGCONC	DRUGPARENT	NUMERIC	8	2.6109771729	2.61
6606	2	1	14:00	Active	DRUGCONC	DRUGMETAB1	NUMERIC	9	0.7410264015	0.74
6906	2	1	14:00	Active	DRUGCONC	DRUGMETAB2	NUMERIC	10	0.2233157158	0.22

Just as it does on the page, this dataset organization takes more space on disk. However, this organization allows for savings in programming time that more than compensate for its size on disk and for any inefficiencies in processing time. Here are the key features of deepfile organization:

- The name of the original dataset and of the original variable move inside each record. That is, much of the metadata becomes data, suitable for processing by ordinary SAS statements. Moving variable names inside the records can also be accomplished by PROC TRANSPOSE, but on a dataset-by-dataset basis; a deepfile takes the same concept even further.
- There is a column, VARSORT, representing the desired sorting order of variables for processing, and you ordinarily sort the deepfile by that column with a higher priority than any other column. This lets you group all of the records for the same dependent variable together, facilitating further statistical processing. Without such a column, it would be necessary for you to sort the deepfile alphabetically by form and variable name; this would also put each variable's data into contiguous records, but the alphabetical order of variables might not be the optimal order for printing or for other processing. Typically, but not necessarily, you will sort the data by other keys within the record groups defined by VARSORT.
- Each record is keyed by the same variables used in the original widefile and by columns identifying which widefile and which variable within the widefile were its source. You might also include the variable label, not just the variable name; it makes the file wider (and hence larger), but it can eliminate the need for retrieving that information later.
- If there is other information that is not a record key but which is frequently needed to organize or classify data, such as the treatment each patient received, you can also include that information in each record. At some point during processing, you may also need to add information such as covariates; you may either include that on the master deepfile (if convenient) or in temporary extractions for special purposes.
- Each variable is represented in both its numeric form (missing in the case of alpha variables), suitable for whatever computations require numeric variables, and by its already-formatted alpha form.
- As implemented here, these data are not in third normal form: there is redundancy in the information in the various columns. Given just the VARSORT variable, SAS data steps could look up the values of DATASET and VARNAME in an indexed data dictionary. However, most procedures would require these theoretically redundant variables (and possibly also a VARLABEL variable) to properly identify variables on print-formatted output, so the standard practice is to include them if the output of standard procedures is to be printed directly without further massaging by DATA steps. There is another important redundancy: in the original widefile database, only one record represented the fact that vital signs were collected at a specific time of a specific day (in addition to representing the value of each sign); however, in the deepfile, that relation is repeated in the keys for each sign that was collected on that occasion. (In the example above, the relation concerning time 8:00 is repeated in records 16, 916, 1816, …, 5416.) Not only do such redundancies increase space and processing time requirements, but in some kinds of databases (those being updated frequently), they increase processing

complexity because the same information must be updated more than once. However, when datasets are used for the kind of statistical analyses discussed here, the code to replicate the redundant information is so straightforward that it does not create any risk to database integrity.

ADVANTAGES OF DEEPFILES FOR ITERATED PROCESSING

If you want to use the MIXED procedure to determine whether the non-key numeric variables depend on DAY, TIME, TREATMENT, or their interaction, and if you want to automate the task of repeating the analysis over the two widefile datasets and the numeric variables within each one, you need to write a macro with nested loops: one level loops over the datasets, while the second loops over the variables within each dataset.

Because the deepfile organization captures the whole study in a single dataset, you might think that all it accomplishes is reducing the need for a nested macro to a need for a single macro. However, you can avoid macros entirely, replacing them with simple by-group logic. In this example, there are two steps: one step consults a data dictionary to see which variables need analysis, while the second step performs the analysis:

```
DATA MIXEDFILE;
   SET DEEPFILE;
   BY VARSORT;
   IF (FIRST.VARSORT) THEN DO;
      SET DATADICT(KEEP=VARSORT ANALYSIS) KEY=VARSORT / UNIQUE;
      RETAIN ANALYSIS;
      END;
   IF (ANALYSIS='MIXED');
RUN;
PROC MIXED DATA=MIXEDFILE;
   BY VARSORT VARNAME;
   CLASS TREATMENT DAY TIME;
   MODEL NUMERICVAL=TREATMENT|DAY|TIME;
   RANDOM PATIENT;
RUN;
```

When this code is run, the printed output and the corresponding datasets do not identify the dependent variable by its original name, such as SYSTOLIC. Instead, they always call the dependent variable NUMERICVAL, and the identification of which numeric variable is being processed is carried in the BY-variables. This is typical of operations on deepfiles: the BY-variables carry information that was originally carried in the variable names and labels in the widefiles. Note that the value of VARSORT may be uninteresting to the ultimate reader of the reports, but including it in the BY statement helps to organize the output, especially if ODS-generated datasets produced by this procedure need to be merged with other datasets later – VARSORT is an excellent, concise, unambiguous merging key. Alternatively, the BY statement in the second step could have been this one:

```
   BY VARNAME NOTSORTED;
```

This has the advantage of leaving VARSORT off the print-formatted output, where it might be considered distracting.

In the example above, the variables to be processed were selected by reference to the data dictionary, prior to creating a separate dataset. However, you can sometimes instead select on properties of DATASET and VARNAME. For example, this statement will select just the drug concentrations for processing:

```
      WHERE (SUBSTR(VARNAME,1,4)='DRUG');
```

If you want to include not only the original variable name but also the original variable label on the output, what do you need to do? There are two ways to handle this. If you're going directly from procedures to print-formatted output, you need to make the deepfile wider by including a VARLABEL field as well as a VARNAME field, including both in the BY statement. Alternatively, if you are going to reorganize and manipulate ODS-generated output datasets before the printing stage, you can leave the variable label off of the deepfile and retrieve it from the data dictionary only when needed. Here is some code illustrating the second approach:

```
ODS LISTING EXCLUDE ALL;
PROC MIXED DATA=MIXEDFILE;
   BY VARSORT VARNAME;
   CLASS TREATMENT DAY TIME;
   MODEL NUMERICVAL=TREATMENT|DAY|TIME;
   RANDOM PATIENT;
   ODS OUTPUT TESTS3=TESTS3;
RUN;
DATA TESTS3PAGE;
   SET TESTS3;
   SET DATADICT(KEEP=DATASET VARLABEL VARSORT) KEY=VARSORT / UNIQUE;
  /* Improve the readability of the by-line */
   LABEL VARSORT='#';
   LABEL VARNAME='Name';
```

```
   LABEL VARLABEL='Measure';
RUN;
PROC PRINT DATA=TESTS3PAGE NOOBS;
   BY VARSORT VARNAME VARLABEL;
   VAR EFFECT NUMDF DENDF FVALUE PROBF;
TITLE3 'Type 3 F tests from PROC MIXED';
RUN;
```

That code results in a condensed report with sections like this:

```
------------- #=5 Name=BODYTEMP Measure=Body temperature, Celsius -------------
                                        Num      Den
                  Effect                 DF       DF     FValue      ProbF
                  TREATMENT               1      869       0.01     0.9149
                  DAY                     4      869       0.16     0.9592
                  TREATMENT*DAY           4      869       1.00     0.4091
                  TIME                    2      869       0.35     0.7047
                  TREATMENT*TIME          2      869       0.48     0.6178
                  DAY*TIME                8      869       1.14     0.3310
                  TREATMENT*DAY*TIME      8      869       0.73     0.6637
```

THE DEEPFILE DATA DICTIONARY

DATA DICTIONARY USES

A deepfile is most useful when used with a properly structured data dictionary, indexed at least on VARSORT and possibly on other ways of identifying a variable, such as the combination of DATASET and VARNAME. Such a dictionary would typically include at least the DATASET, VARNAME, and VARSORT fields shown above, along with a VARLABEL field. However, it can also contain other information useful at any stage of data management or statistics. For example, it could contain an ANALYSIS column with values such as "CHISQUARE" or "MIXED", allowing you to quickly select only those variables suitable for a specific planned analysis. Alternatively, it could contain columns for MINVALUE and MAXVALUE, allowing you to automate the task of reporting and investigating out-of-range values, as in this example:

```
DATA _NULL_;
   SET DEEPFILE;
   WHERE (NUMERICVAL^=.);
   BY VARSORT;
   FILE PRINT;
   IF (FIRST.VARSORT) THEN DO;
      SET DATADICT(KEEP=VARSORT VARLABEL MINVALUE MAXVALUE VARFORMAT)
         KEY=VARSORT / UNIQUE;
      RETAIN VARSORT VARLABEL MINVALUE MAXVALUE VARFORMAT;
      LINE='Out-of-range values for '||TRIM(VARLABEL)||
           ' (range is '||PUTN(MINVALUE,VARFORMAT)||' to '||
                         PUTN(MAXVALUE,VARFORMAT)||')';
      PUT / LINE;
      END;
   IF ((NUMERICVAL<MINVALUE) OR (NUMERICVAL>MAXVALUE)) THEN DO;
      LINE='   Patient '||Put(Patient,2.)||', Day '||PUT(DAY,1.)||
           ', Time '||PUT(TIME,TIME5.)||': '||PUTN(NUMERICVAL,VARFORMAT);
      PUT LINE;
      END;
TITLE3 'Out-of-range numeric variables';
RUN;
```

With only those 21 lines of code and no macros, you can produce a list of out-of-range values, formatted like this:

```
Out-of-range values for Breaths per minute (range is  7 to 20)
Patient  3, Day 5, Time 18:00: 33
Patient  4, Day 1, Time  8:00: 32
Patient  5, Day 1, Time 18:00: 36
Patient  8, Day 3, Time 18:00: 35
```

If you generally find the output of procedures to be organized in a way that does not meet your needs, and you use DATA _NULL_ steps or procedures such as PRINT or REPORT to format output datasets for display, you can also use the data dictionary to customize this output process. For example, if different variables are to be reported to different numbers of decimal places within the same column of a single listing, you can let the data dictionary contain the desired output format in a column called VARFORMAT and containing values such as "10.3", "DATE9.", or "SEXFMT.". In the following example, the means of those continuous numeric variables which were analyzed using PROC MIXED, above, are converted to character variables using formats specific to each variable:

```
PROC MEANS DATA=MIXEDFILE NOPRINT;
   BY VARSORT;
   VAR NUMERICVAL;
   OUTPUT OUT=MEANFILE2 MEAN=MEAN;
RUN;
/* Represent the grand means with their proper
   variable labels and data formats */
DATA MEANFILE3(KEEP=VARLABEL GRANDMEAN);
   SET MEANFILE2;
   SET DATADICT(KEEP=VARSORT VARLABEL VARFORMAT) KEY=VARSORT / UNIQUE;
   ATTRIB GRANDMEAN LENGTH=$10 LABEL='Grand mean';
   LABEL VARLABEL='Dependent variable';
  /* Temporarily left justify the character version */
   GRANDMEAN=LEFT(PUTN(MEAN, VARFORMAT));
  /* If the grand mean has a decimal point, move it to column 6;
     otherwise, move the last digit to column 5 */
   J=INDEX(GRANDMEAN,'.');
   IF (J=0) THEN
     /* Follow it with an imaginary decimal point */
      J=LENGTH(TRIM(GRANDMEAN))+1;
   GRANDMEAN=REPEAT(' ',5-J) || GRANDMEAN;
   FORMAT GRANDMEAN $CHAR10.;
RUN;
PROC PRINT DATA=MEANFILE3 LABEL;
   VAR VARLABEL GRANDMEAN;
RUN;
```

This code produces the following listing, in which the formatting changes from line to line:

Obs	Dependent variable	Grand mean
1	Systolic blood pressure	121
2	Diastolic blood pressure	79
3	Heart rate, BPM	60
4	Breaths per minute	10
5	Body temperature, Celsius	36.8
6	Parent compound concentration	1.83
7	Metabolite 1 concentration	0.53
8	Metabolite 2 concentration	0.19

DATA DICTIONARY CONSTRUCTION

The process of constructing a data dictionary typically starts with the extraction and organization of all of the metadata you already have. Such metadata may be present only in the widefile definitions (from which you can extract it using PROC CONTENTS), or it may be present in a non-SAS format such as a spreadsheet. After that, the completion of the data dictionary requires consolidating that information with other information sources, such as ranges of valid values and intended analyses. If the consolidation requires much manual keying, you may find it easiest to perform it outside of SAS, on a spreadsheet where there are tools and cell formulas to automate some of the work, and then to re-import the result into SAS. Finally, the data dictionary should be both sorted and indexed by VARSORT. For the current study, this would be a reasonable set of data dictionary fields:

VIEWTABLE: Work.Datadict

	DATASET	VARNAME	VARLABEL	VARTYPE	VARSORT	VARFORMAT	MINVALUE	MAXVALUE	ANALYSIS
1	VITALS	SYSTOLIC	Systolic blood pressure	NUMERIC	1	3.	100	150	MIXED
2	VITALS	DIASTOLIC	Diastolic blood pressure	NUMERIC	2	3.	70	100	MIXED
3	VITALS	HEARTRATE	Heart rate, BPM	NUMERIC	3	3.	45	90	MIXED
4	VITALS	BREATHRATE	Breaths per minute	NUMERIC	4	2.	7	20	MIXED
5	VITALS	BODYTEMP	Body temperature, Celsius	NUMERIC	5	4.1	36.5	39	MIXED
6	VITALS	SKIN	Skin response	NUMERIC	6	SKINCODE.	0	1	CHISQUARE
7	VITALS	COMMENT	Staff comment	ALPHA	7	$20	.	.	
8	DRUGCONC	DRUGPARENT	Parent compound concentration	NUMERIC	8	5.2	0.05	10	MIXED
9	DRUGCONC	DRUGMETAB1	Metabolite 1 concentration	NUMERIC	9	5.2	0.05	10	MIXED
10	DRUGCONC	DRUGMETAB2	Metabolite 2 concentration	NUMERIC	10	5.2	0.05	10	MIXED

The data dictionary construction process can also anticipate derived variables to be added to the deepfile using DATA steps. For example, you might expect to later add the variable MAP, or Mean Arterial Pressure, which is a weighted average of SYSTOLIC and DIASTOLIC. In the deepfile above, that variable can have VARSORT=2.5, coming after the existing diastolic pressure but before the heart rate. There is no need for the values of VARSORT to be consecutive integers: any unique numbers that sort properly will do. Sometimes it is convenient, for example, to use 4-digit values of VARSORT, where the first two digits identify the original dataset name and the last two sort the variables within that dataset. If such derived variables are added, it is useful (at least for documentation, if not for

processing) to include a data dictionary variable called SOURCE, with values such as "Data entry" and "Derived", to indicate which variables are added in subsequent data steps.

GENERATING DEEPFILES FROM WIDEFILES

If your data dictionary contains sufficient information, it can be used to automatically generate the SAS code for converting an existing set of widefiles to a deepfile. When that code is executed, each widefile becomes one portion of the deepfile, and you can take two approaches to combining these portions into a consolidated deepfile. You can generate all of the separate partial deepfiles first, giving each a separate name (and saving those names to a temporary dataset), ending the code with procedures to merge and sort the result. Alternatively, you can keep re-using the same name for the temporary deepfile created by each widefile, appending it to an originally-empty deepfile immediately after creating it, again ending with a sort of the final result. Both the code generator and the code it generates are too large to show here in their entirety, but the generated code consists primarily of sections organized like the following fragment:

```
DATA PARTIAL_DEEPFILE (KEEP=PATIENT DAY TIME TREATMENT
                           DATASET VARNAME VARTYPE VARSORT
                           NUMERICVAL ALPHAVAL);
   LENGTH PATIENT DAY TIME 4 TREATMENT $8
          DATASET VARNAME $16 VARTYPE $8 VARSORT 4
          NUMERICVAL 4 ALPHAVAL $40;
   FORMAT TIME TIME5.;
   SET VITALS;
   DATASET="VITALS";
  /* DAY AND TIME ARE ALREADY ON THE DATASET */
   SET DECODING KEY=PATIENT / UNIQUE; /* GET TREATMENT */

   VARNAME="SYSTOLIC";
   VARTYPE="NUMERIC";
   VARSORT=1;
   NUMERICVAL=SYSTOLIC;
   ALPHAVAL=PUTN(SYSTOLIC,"3.");
   OUTPUT;

   VARNAME="DIASTOLIC";
   VARTYPE="NUMERIC";
   VARSORT=2;
   NUMERICVAL=DIASTOLIC;
   ALPHAVAL=PUTN(DIASTOLIC,"3.");
   OUTPUT;
/* Code to output other variables is not shown here */
RUN;
PROC APPEND BASE=DEEPFILE DATA=PARTIAL_DEEPFILE;
RUN;
```

Writing large amounts of such code manually is a mind-numbing activity, but it is quite straightforward to write code that loops through the records of the data dictionary, writes the lines DATA through SET whenever FIRST.DATASET, writes a block of lines from VARNAME= through OUTPUT for each data dictionary record, and writes the RUN, PROC APPEND, and RUN lines whenever LAST.DATASET. You can then automatically %INCLUDE the generated code in the execution of the same program, immediately after the code that generates it.

ADDING DERIVED VARIABLES TO DEEPFILES

Derived variables are typically of two kinds: summaries of the same variable over different occasions, or summaries across different variables on the same occasion.

A deepfile is usually sorted by VARSORT, which is the best order for repeating the same univariate procedure for different dependent variables. This organization is suitable for creating a summary over occasions, especially if you are willing to let the missing value for a key represent "averaged over all values of the key", just as the MEANS procedure does when dealing with a class variable. Here is the code to average each of the continuous vital signs measures over all of the times during the day:

```
PROC MEANS DATA=MIXEDFILE NOPRINT NWAY;
   WHERE (DATASET='VITALS');
   BY VARSORT;
   CLASS PATIENT DAY;
   VAR NUMERICVAL;
   OUTPUT OUT=MEANFILE2 MEAN=MEAN;
RUN;
```

```
DATA SUPPLEMENT1;
   SET MEANFILE2(DROP=_FREQ_ _TYPE_ RENAME=(MEAN=NUMERICVAL));
   TIME=.;
   SET DATADICT(KEEP=VARSORT VARNAME VARFORMAT) KEY=VARSORT / UNIQUE;
  /* Note that the default format does not allow any extra decimal
     places for the computed mean. */
   ALPHAVAL=PUTN(NUMERICVAL,VARFORMAT);
   DROP VARFORMAT;
  /* Warning: the resulting dataset SUPPLEMENT1 does not have a
     structure that is fully consistent with the data dictionary.
     In particular, there are no records for SKIN or COMMENT when
     TIME=.  Depending on the use to be made of the file, this may
     or may not be a problem. Here, we assume that it might be a
     problem, and we keep the revised deepfile separate. */
RUN;
DATA REVISED_DEEPFILE_1;
   MERGE DEEPFILE SUPPLEMENT1;
   BY VARSORT PATIENT DAY TIME;
RUN;
```

Note that this code sorts all of the averages (represented by TIME=.) ahead of the values from which they were derived. Alternatively, you may wish to have them sort to the end, by adopting a convention such as TIME=23:99 for the end-of-day average (and you can always attach a format that prints such a value as the word "MEAN").

The standard organization by VARSORT, however, is inappropriate for creating new variables that are derived from information spread over different variables within an occasion, rather than over different occasions of the same variable. It may be inappropriate to create such variables while processing the original widefiles, perhaps because the widefiles are being reserved for original data entry before any computed variables are added. In such a case, it may be necessary to create the new variable from the DEEPFILE representation. Rather than resorting the whole DEEPFILE to place data values collected at the same time into adjacent records, where they may be combined conveniently, it is better to sort only the required subset of the data:

```
PROC SORT DATA=DEEPFILE OUT=ENHANCED1;
   WHERE (VARSORT IN (1,2));
   BY PATIENT DAY TIME VARSORT;
RUN;
DATA ENHANCED2;
   SET ENHANCED1;
   BY PATIENT DAY TIME VARSORT;
   IF (VARSORT=1) THEN
      SYSTOLIC_SAVE=NUMERICVAL;
   RETAIN SYSTOLIC_SAVE;
   DROP SYSTOLIC_SAVE;
   IF (VARSORT=2) THEN DO;
      NUMERICVAL=(SYSTOLIC_SAVE+2*NUMERICVAL)/3;
      ALPHAVAL=PUT(NUMERICVAL,3.);
      VARSORT=2.5;
      VARNAME="MAP";
      OUTPUT;
      END;
RUN;
PROC SORT DATA=ENHANCED2;
   BY VARSORT PATIENT DAY TIME;
RUN;
DATA NEW_DEEPFILE;
   SET DEEPFILE ENHANCED2;
   BY VARSORT PATIENT DAY TIME;
RUN;
```

Note that this code, unlike the previous code, creates records with values of VARSORT and VARNAME that were not in the original data dictionary. Therefore, you need to either create an enhanced data dictionary to correspond to these new variables or else include records for them in the original data dictionary but filter them out (using WHERE statements) when they're irrelevant.

ADVANTAGES OF DEEPFILES FOR EVENT-ORDER REPORTING

Although we often like to see data organized by domain (for example, all patients' vitals signs for all occasions in a single table), there are times when it is useful to organize data by the time that it is collected, interspersing different

forms (records from different original widefile datasets) on an as-needed basis. In the pharmaceutical industry, this mimics the organization of a case report form, or CRF, in which pages in a binder are completed front-to-back, and instances of the same form keep recurring on different occasions when the form is to be used. Using a deepfile and its data dictionary, it is easy to produce a report that is structured in this way. The report is not concise, and one might not wish to convert it to hardcopy form, but it provides a useful case narrative for on-screen review. Here is the code to produce such a listing, generating a separate ASCII file for each patient:

```
FILENAME CRFFILE 'TEMPORARYNAME.TXT';
PROC SORT DATA=DEEPFILE(WHERE=(TIME^=.)) OUT=CRFORDER;
   BY PATIENT DAY TIME VARSORT;
RUN;
DATA _NULL_;
   LENGTH CURRENTFILENAME $60;
   RETAIN FILENAMES 0 CURRENTFILENAME;
   SET CRFORDER;
  /* Note that VARSORT groups the variables of each dataset together,
     but the dataset names are not necessarily in alphabetical order */
   BY PATIENT DAY TIME DATASET NOTSORTED;
   IF (FIRST.PATIENT) THEN DO;
     /* Start a new ASCII file for each patient */
      FILENAMES=FILENAMES+1;
      CURRENTFILENAME="CRF-STYLE LISTING " ||PUT(FILENAMES,Z2.) || ".TXT";
      END;
   FILE CRFFILE FILEVAR=CURRENTFILENAME;
   IF (FIRST.DATASET) THEN
      PUT / 'Patient ' PATIENT '  Day ' DAY '  Time '
            TIME TIME5. '   Form ' DATASET;
  /* Retrieve the variable label */
   SET DATADICT(KEEP=VARSORT VARLABEL) KEY=VARSORT / UNIQUE;
  /* Right-justify the label prior to printing */
   VARLABEL=RIGHT(VARLABEL);
  /* The information is already formatted properly for printing, but note
     that we need format $CHAR40. to prevent the text from left-justifying.
     If the dataset structure allows for many fields that are only rarely
     used, it is useful to output this value only if ALPHAVAL^=''. */
   PUT VARLABEL $CHAR40. +1 ALPHAVAL;
RUN;
```

This is an excerpt from the resulting listing:

```
Patient 2   Day 1   Time 13:00   Form VITALS
                Systolic blood pressure 112
               Diastolic blood pressure 83
                        Heart rate, BPM 51
                     Breaths per minute 10
              Body temperature, Celsius 37.0
                          Skin response Normal
                          Staff comment

Patient 2   Day 1   Time 14:00   Form DRUGCONC
          Parent compound concentration 2.61
             Metabolite 1 concentration 0.74
             Metabolite 2 concentration 0.22
```

REORDERING VARIABLES

A recurring question among SAS users is how to conveniently reorder variables within a dataset. The reasons for reordering are many, but two of them are of particular interest: to allow convenient reference to all relevant variables with arrays or double-hyphen notation (e.g., "SYSTOLIC--BODYTEMP"), and to control the order in which variables appear on printed reports. Using the deepfile approach, you can reorder variables quite easily: you simply replace the values of VARSORT (not necessarily with integers), modify the dictionary accordingly, and resort the deepfile. (If the modified deepfile is to be processed only by procedures, not by datasets, you may find it sufficient to reindex it instead of to resort it.) When procedures process the deepfile, they will output results in the appropriate order. When any new widefiles or ASCII representations in widefile format are generated from the deepfile, their internal variable order will also follow the new values of VARSORT.

GENERATING WIDEFILES FROM A DEEPFILE

Some SAS tasks require data in a widefile organization. For example, any of the statistical procedures (such as CORR, REG, or FACTOR) that construct correlation matrices require all of the variables in the matrix to be part of the same record. In many cases, the easiest way to extract and reorganize a subset of data is to manually code SORT and TRANSPOSE steps. In this example, we start with the modified deepfile that includes the new variable, MAP, computed earlier:

```
PROC SORT DATA=NEW_DEEPFILE OUT=CORRFILE1;
   WHERE ((VARSORT>=1) AND (VARSORT<=5));
   BY PATIENT DAY TIME VARSORT;
RUN;
PROC TRANSPOSE DATA=CORRFILE1 OUT=CORRFILE2;
   BY PATIENT DAY TIME;
   VAR NUMERICVAL;
   ID VARNAME;
RUN;
PROC CORR DATA=CORRFILE2;
   VAR SYSTOLIC--BODYTEMP;
TITLE3 'Overall correlations among vital signs';
RUN;
```

The resulting dataset is then in a widefile format. Note that, because you are back to a widefile format, you need to explicitly code the names of the variables to be included in the correlation. If you wanted to compute this kind of correlation for the numeric variables in each of many original widefiles, the deepfile organization might not eliminate your need to write macros. Each of your separate calls to PROC CORR would involve a different number of dependent variables, and it would be impractical to let a fixed set of variable names (such as "NUMERICVAL1" through "NUMERICVAL6") substitute for different collections of dependent variables, each collection of a different size. This is unlike the earlier situation with PROC MIXED, where there was always only a single dependent variable in use at any one time, and therefore the structure of the code did not need to change for different datasets or variables.

Note that the process for creating widefiles described above works easily only when you want to output only numeric or only text variables to a single widefile for further processing. If you require a dataset of mixed numeric and text values, or if you wish to automatically convert a deepfile to an entire library of widefiles, then it is necessary for you to write a code generator based on the data dictionary, more or less the reverse of the code generator described above. However, if you want to convert a deepfile to the ASCII representation of widefiles, that process is much easier. (One reason that it's easier is that a SAS DATA step can generate the names of multiple ASCII output files at run time, while it requires the names of any SAS datasets to be known at the time the step is compiled.) Here, for example, is part of the code that converts an entire deepfile to tab-separated ASCII that can be imported immediately by a spreadsheet:

```
PROC SORT DATA=DEEPFILE OUT=FORMORDER;
   BY DATASET PATIENT DAY TIME VARSORT;
RUN;
FILENAME TABFILE 'TEMPORARYNAME.TXT';
DATA _NULL_;
   LENGTH CURRENTFILENAME $60;
   RETAIN CURRENTFILENAME;
   SET FORMORDER;
   BY DATASET PATIENT DAY TIME VARSORT;
   RETAIN TAB '09'X;
   IF (FIRST.DATASET) THEN
     /* Change the ASCII file name for each dataset */
      CURRENTFILENAME="DATASET " ||TRIM(DATASET) || ".TXT";
   FILE TABFILE FILEVAR=CURRENTFILENAME;
  /* Output row identification fields at the beginning of each row */
   IF (FIRST.TIME) THEN
      PUT PATIENT +(-1) TAB $1.
          DAY     +(-1) TAB $1.
          TIME    TIME5. TAB $1. @;
   PUT ALPHAVAL +(-1) TAB $1. @;
   IF (LAST.TIME) THEN
      PUT;
RUN;
```

The SAS export wizard can also write code that outputs a widefile as a delimited file. However, that code generated by SAS includes a separate PUT statement for each original variable. If you want to do something unusual, such as replacing each numeric missing value with the "~" character on the output file, you need to make changes in

numerous places. Using the deepfile approach, as above, you need to make the change in only one place, just before the end of the step:

```
IF (NUMERICVAL=.) THEN
   ALPHAVAL='~';
PUT ALPHAVAL +(-1) TAB $1. @;
```

WHY ARE DEEPFILES SO POWERFUL?

When we want to process a collection of dependent variables, stored under different names in different datasets, SAS needs to refer to a variety of different sources to find the information. The dataset names are stored in the disk storage system's various tables of contents and copied, for convenient reference, to SASHELP.VTABLE. The variable (column) names within each dataset are stored in the dataset header and copied, for convenient reference, to SASHELP.VCOLUMN. The dataset and variable names are metadata. The actual data values are stored inside the datasets. Within SAS, almost all references to data values must be made by executing code, and within that code each variable must be referenced by name or as a member of an array. (The latter can be inconvenient, not least because numeric and text variables may not be mixed in the same array.) Automating the production of that code requires moving back and forth between the metadata domain and the data domain, using separate sets of tools: in general, the macro system can perform computations on metadata, while DATA steps and procedures can perform computations on data.

Deepfiles are a form of data organization that bring the most useful parts of the metadata world, the names of datasets and variables, into the same processing domain as the data values themselves. This allows for much useful processing to occur inside directly-coded DATA and procedure steps, rather than within indirectly-coded steps generated by the macro language. Even when deepfile data dictionaries are used to generate code, as typically happens when converting widefiles to deepfiles, debugging the code generator, which is after all an ordinary DATA step, may be easier than debugging macros. While macros are largely limited to outputting code and to leaving macro variables in memory, DATA steps processing dictionaries and deepfiles can generate a wide range of diagnostic products – new SAS datasets, text for the log, text for the output listing, and text files to be executed later – greatly easing the burden of developing and debugging code.

CONCLUSION

Reorganizing a collection of datasets as a single master file in deepfile format, with an associated data dictionary, can greatly simplify the task of coding many processing steps. For datasets that are small enough for programmers' time to be the resource most worth optimizing, the resulting costs in disk space and processing time are well repaid in personal productivity.

ACKNOWLEDGMENTS

The author appreciates the suggestions that Cindy Bartlett and Karen Kaplan made when reviewing a draft of this paper and the accompanying sample program.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Howard L. Kaplan
Ventana Clinical Research Corporation
340 College Street, Suite 400
Toronto, Ontario Canada M5T 3A9
h.kaplan@ventana-crc.com

A sample program (about 20 KB) demonstrating these techniques in more detail is available from the author or from the downloads area of http://www.ventana-crc.com.