

Runtime fun time: Vertical data reporting using runtime formats

Michael Auld, London, UK

ABSTRACT

With the industry-wide move towards standardisation using the CDISC SDTM, the perceived wisdom is to create more vertical data structures. When it comes to creating Derived data sets, many argue that this approach is incompatible with parameter-based data, for example Vital Signs, where the precision and number of decimal places varies from parameter to parameter. An approach adopted by many is to create separate variables for each parameter - thus creating a wide horizontal data set.

This transposition of the data is a backwards step in the ultimate goal of the CDISC consortium, which is to harmonise and integrate the SDTM and the ADAM standards.

This paper identifies a solution to this problem by retaining the SDTM vertical structure and using the under-utilised and often forgotten SAS® runtime format facility. In this way, parameter-based data like Laboratory results can be analysed and displayed to the correct level of precision in which they were collected

INTRODUCTION

This paper promotes the implementation of vertical Analysis data sets, and offers run-time formats as a solution to retaining the metadata about each parameter without having to transpose the data to a horizontal structure. Knowledge of the CDISC SDTM data structure is a help but not essential.

PARAMETER-BASED DATA – AN ABSTRACT VIEW

How difficult to understand is clinical data? Is there really such a great difference between ECG results and questionnaire data? Concomitant Medication and Adverse Events? Dictionaries and Formats and Code Lists? Sometimes it is possible to become too immersed in a specific individual set of data (referred to as a data domain) to note the symmetries that exist across differing sets of unrelated data domains.

When creating the SDTM, CDISC have grouped data domains into 3 main groups (or observation classes) – Interventions, Events or Findings. Parameter-based data (which this paper will focus on) falls into the category of “Findings” (like most clinical data), though this does not rule out the likelihood of using a similar strategy in other areas.

Whilst this is not an exercise in teaching good programming practice, it is now accepted that in reporting clinical trials it is best if as much programming as possible is done early on, through the creation of Derived data sets. This analysis data can then be used in programs that generate the table, listing and figures used in the Clinical Study Report.

This has two advantages – it ensures that a consistent approach is used in reporting a particular data domain by ensuring that variables are derived (and maintainable) in one place only. Secondly, the complexity of the TLF programs are reduced, opening up opportunities for increased automation and reduction in QC overheads

DATA HARMONISATION: THE CASE FOR VERTICAL DATA STRUCTURES

In my view, the design of the analysis data set has failed if the subsequent table listing or figure programs become too long or complicated, or has to be post-processed (for example by using transposition) in some way. However, it may be argued that transformation may be necessary in order to present the data as specified in Tables, Listings and Figures Shells. It must be imperative therefore that to achieve harmonisation in database design, there must be similar movement in standardisation of presented outputs.

PhUSE 2007

Consider a pre-SDTM model for raw Vital Signs:

STUDY A, VITAL SIGNS

USUBJID	VISIT	HEIGHT	WEIGHT	PULSE
00001	SCREEN	183	88.2	63
00002	SCREEN	178	81.4	70
00003	SCREEN	163	68.0	72

Let us call this Study A. When creating derived analysis data, this appears to be a quite straightforward 1-to-1 mapping, each vital signs parameter occupying a separate variable. However, for the same compound, Study B also wishes to capture BMI information:

STUDY B VITAL SIGNS, NOW WITH ADDED BMI

USUBJID	VISIT	HEIGHT	WEIGHT	BMI	PULSE
00001	SCREEN	183	88.2	22	63
00002	SCREEN	178	81.4	23	70
00003	SCREEN	163	68.0	24	72

The negative side to adding this extra variable horizontally means that we are no longer able to re-use code from Study A to Study B without modification when creating our analysis Vital Signs dataset. The programmer must remember to add the additional BMI variable not only in the derived dataset program, but also in subsequent Tables, Listings and Figures programs where appropriate. The situation becomes untenable as the parameterisation increases, for example in a laboratory dataset, where the number of columns can be expected to vary from study to study.

LABORATORY DATA – HORIZONTAL STRUCTURE

USUBJID	VISIT	POTASSIUM	PLATELETS	RBC	GAMMAGT
00001	SCREEN	4.3	316	4.17	16
00002	SCREEN	5.1	350	4.32	17
00003	SCREEN	4.2	312	4.89	18

Employing a vertical structure allows for easier addition of the study optional items to be incorporated without harming the database design, just by adding an additional observation to the dataset:

STUDY B VITAL SIGNS – VERTICAL STRUCTURE

USUBJID	VISIT	CATEGORY	RESULT
00001	SCREEN	HEIGHT	183
00001	SCREEN	WEIGHT	88.2
00001	SCREEN	BMI	22
00001	SCREEN	PULSE	63
00002	SCREEN	HEIGHT	178
00002	SCREEN	WEIGHT	81.4
00002	SCREEN	BMI	23
00002	SCREEN	PULSE	70

PhUSE 2007

The above solution is even more flexible if we extend the principle of symmetry across data domains. Re-use of code becomes a greater possibility, and so code that can produce summary tables for laboratory results can be re-used for vital signs.

By reducing each data domain to a granular level, then we have a consistent approach across all domains. The SDTM implementation guide gives guidance for creating new domains, but the principles can equally be applied to the design of already published standard domains. The preferred approach adopted in SDTM is a vertical, parameterised data structure, consisting of categories, sub-categories and results for a subject at a particular timepoint.

ADDRESSING VARYING LEVELS OF PRECISION

Having identified a flexible yet stable structure for storing the data, we are left with a considerable drawback. The precision (or more bluntly, the number of decimal places), to which the results have been captured varies from parameter (or category) to parameter.

If the vertical structure has been used in the raw clinical database, (for instance, laboratory data), this has often been resolved by storing the numeric data as a character string, with a numeric version provided with a fixed precision.

LABORATORY DATA – VERTICAL STRUCTURE, WITH CHARACTER DATA WITH VARYING PRECISION AND NUMERIC DATA WITH FIXED PRECISION

USUBJID	VISIT	PARAMETER	LBSTRESC	LBSTRESN
00001	SCREEN	GAMMA GT	16	16.000
00001	SCREEN	POTASSIUM	4.3	4.300
00001	SCREEN	RED BLOOD CELLS	4.17	4.170
00001	SCREEN	PLATELETS	316	316.000
00002	SCREEN	UREA	3.3	3.300
00002	SCREEN	LEUKOCYTES	6.73	6.730
00002	SCREEN	CREATININE	79	79.000
00002	SCREEN	HCT	0.378	0.378

The numeric data can still be presented reasonably sensibly if a best. format is applied as an attribute, but against each parameter the level of precision recorded with the value is hidden. What is really required is raw parameter based data to be provided with the precision level associated with each value.

“There is no place to store individual attributes for values of VSTESTCD in the standard metadata model.”
(CDISC ADAM 2.0)

This last quote represents the challenge of this paper. A short-term strategy in designing a reporting data warehouse was to implement a set of standards that drew on SDTM naming conventions and structure, but apply the design principles of AdAM. Surely a transformation of the SDTM data was an obstacle to the CDISC long-term strategic plan of the harmonisation of SDTM and AdAM models?

Luckily, SAS offers a solution that avoids the suggested data transposition of the SDTM by using format functions (PUTN, PUTC, INPUTN, INPUTC) that change with each observation – instead of being fixed. With this knowledge, the attribute information about the results can be stored as an additional variable for each individual observation. With this in mind, I set about adding a precision variable to the existing vertical laboratory data.

PhUSE 2007

LABORATORY DATA – VERTICAL STRUCTURE, WITH CHARACTER DATA WITH VARYING PRECISION, NUMERIC DATA WITH FIXED PRECISION, AND DATA VALUE PRECISION

USUBJID	VISIT	PARAMETER	LBSTRESC	LBSTRESN	PARAMDP
00001	SCREEN	GAMMA GT	16	16.000	0
00001	SCREEN	POTASSIUM	4.3	4.300	1
00001	SCREEN	RED BLOOD CELLS	4.17	4.170	2
00001	SCREEN	PLATELETS	316	316.000	0
00002	SCREEN	UREA	3.3	3.300	1
00002	SCREEN	LEUKOCYTES	6.73	6.730	2
00002	SCREEN	CREATININE	79	79.000	0

Unfortunately, this precision data, which could easily have been provided by the central laboratory – or if paper-based CRFs used then straight from the CRF page, has been derived per parameter by some pre-processing of the clinical data. It was not supplied with the raw data as standard. It is my view that this should be a long-term goal of data providers.

To derive the number of decimal places the following code maybe used:

```

**prepare to get decimal places by parameter;
data preDP;
  length PARAMDP 8;
  set phuse.lab;
  ** Only relevant for reporting numeric data;
  if not missing(LBSTRESN) then do;
    **apply best. format to ensure all decimal places are visible, and write to a string;
    valc = put(LBSTRESN,best.);
    decPoint = index(valc,"."); **locate the decimal point;
    **count number of characters after the decimal point;
    if decPoint then PARAMDP = length(trim(left(substr(valc,decpoint+1))));
    else paramdp = 0;
  end;
  drop valc decPoint;
run;

**now get the maximum number of decimal places recorded for each parameter;
proc means data=preDP nway noprint;
  class LBtestCD;
  var paramDP;
  output out=DPsum max=;
run;

***merge back to the original data;
proc sql noprint;
  create table phuse.labDP as
  select Lab.*,
         case
           when missing(DPsum.paramDP) then " "
           else put(DPsum.paramDP,best.)
         end as LBRESDP
  from phuse.LAB left join DPsum
  on Lab.LBtestCD eq DPsum.LBtestCD
  order by usubjid, visit, LBcat, LBtestCD;
quit;

```

PhUSE 2007

PROC CONTENTS BEFORE ADDED DP

The DATASETS Procedure

Data Set Name	PHUSE.LAB	Observations	372
Member Type	DATA	Variables	7
Engine	V9	Indexes	0
Created	Tue, Sep 25, 2007 09:16:34 PM	Observation Length	184
Last Modified	Tue, Sep 25, 2007 09:16:34 PM	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	WINDOWS_32		
Encoding	wlatin1 Western (Windows)		

Engine/Host Dependent Information

Data Set Page Size	16384
Number of Data Set Pages	5
First Data Page	1
Max Obs per Page	88
Obs in First Data Page	88
Number of Data Set Repairs	0
File Name	C:\phuse07\lab.sas7bdat
Release Created	9.0101M3
Host Created	XP_PRO

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Label
1	DUMMYTRT	Num	8	DUMMYTRT.	Treatment Variable
4	LBCAT	Num	8	8.	Laboratory type
7	LBSTRESC	Char	50	\$50.	Result (text)
7	LBSTRESN	Num	8	BEST.	Result (numeric)
5	LBTESTCD	Char	47	\$47.	Parameter name
2	USUBJID	Char	9	\$9.	Subject identifier (centre_subject)
3	VISIT	Char	50	\$50.	Visit name

PROC CONTENTS AFTER DP ADDED

The DATASETS Procedure

Data Set Name	PHUSE.LABDP	Observations	372
Member Type	DATA	Variables	8
Engine	V9	Indexes	0
Created	Tue, Sep 25, 2007 09:20:33 PM	Observation Length	192
Last Modified	Tue, Sep 25, 2007 09:20:33 PM	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	YES
Label			
Data Representation	WINDOWS_32		
Encoding	wlatin1 Western (Windows)		

Engine/Host Dependent Information

Data Set Page Size	16384
Number of Data Set Pages	5
First Data Page	1
Max Obs per Page	85
Obs in First Data Page	75
Number of Data Set Repairs	0
File Name	C:\phuse07\labdp.sas7bdat
Release Created	9.0101M3
Host Created	XP_PRO

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Label
1	DUMMYTRT	Num	8	DUMMYTRT.	Treatment Variable
4	LBCAT	Num	8	8.	Laboratory type
8	LBRESDP	Num	8		
6	LBSTRESC	Char	50	\$50.	Result (text)
7	LBSTRESN	Num	8	BEST.	Result (numeric)
5	LBTESTCD	Char	47	\$47.	Parameter name
2	USUBJID	Char	9	\$9.	Subject identifier (centre_subject)
3	VISIT	Char	50	\$50.	Visit name

WHAT IS BINDING – AN ASIDE

When a SAS program is submitted, the code is syntax checked and then translated into machine code (compiled). As part of this process, SAS creates a descriptor information object, which creates and maintains information on variable attributes – names, labels, data types and formats. Generally this is bound (or assigned) in the compile-time phase, and therefore remains fixed for the duration of the data step execution.

“Binding time” is a concept common to programming languages (lately object-oriented) and is defined as the time at which the variable and its value are bound together (Thimbleby 1988).

In relation to SAS, Macro Language statements are examples of early binding – having been resolved and executed before the data step is compiled or parsed.

Exceptions to the rule are nested macros – which have to be executed each time they are encountered, which is why this is mainly frowned upon; and pseudo data step commands accessible via %SYSFUNC; in addition to SAS macro, the run-time format facility are all examples of late binding – in that the attributes of the variables in the descriptor information are not fully translated until later in the program execution process.

Compile-time formats are therefore locked in place in the descriptor information. In relation to the laboratory data set above, this means that there is no variability for each value of LBSTRESN. Although the raw data is in the preferred vertical, parameterised structure, and indeed the observations can be read and viewed reasonably consistently, the use of the best. format is not enough to prevent inaccurate observations like observation 3. giving a false impression of the precision of the data collected. Also, when reporting summary statistics, employing a static number of decimal places may be fine for most parameters, but also is misleading (and perhaps unsuitable) for others.

REPORTING THE DATA

At first the benefits of this methodology may not be obvious. You cannot take advantage of the run-time format when listing the data, as the PUTN function is not compatible with standard SAS procedures – it is only accessible via the data step. Therefore, to provide a listing of the numeric variable in combination with a run-time format would require an intermediate data step to create a character variable for reporting. However, this is unnecessary as the data already exists in the character-typed LBSTRESC variable.

The flexibility of using run-time formats is realised when reporting summary statistics tables. The first step is to use PROC SUMMARY to generate the summary statistics:

```
proc means data=phuse.labdp nway noprint missing;
  var LBSTRESN;
  class DUMMYTRT VISIT LBCAT LBTESTCD LBRESDP;
  output out=meanLAB n= mean= stddev= min= max=/autoname;
run;
```

Note that amongst the class variables the LBRESDP variable also needs to be included.

This can then be used to create display formats at run-time for each statistic to be reported. The “Round-off” rule is summarised below for selected statistics:

Statistic	Decimal places to add
MINIMUM	0
MAXIMUM	0
MEDIAN	0
MODE	0
MEAN	+1
STANDARD DEVIATION	+2

Creating the display formats can then be achieved in the following data step:

PhUSE 2007

```
data statLAB;
  length restext stattxt fmt $12;
  set meanLAB;
  array resarr(5) LBSTRESN_n LBSTRESN_mean LBSTRESN_stddev LBSTRESN_min LBSTRESN_max;
  array statlist (5) $ _TEMPORARY_ ("N" "MEAN" "STDDEV" "MIN" "MAX");
  ***Creation of the second array is optional - and included in code to aid clarity.;
  ***You may wish to identify the statistics in the select statement by;;
  *** using macro variables, or a value format;
  ***It is inadvisable to only refer to the index of the array in the select;
  ***This makes the code less self-documenting and difficult to maintain;

  trtlabel = put(dummytrt,dummytrt.);

  ***Now create an observation row for each statistic to be reported;
  do statord = 1 to dim(resarr);
    stattxt = statlist(statord);
    if not missing(resarr(statord)) then select(stattxt);
      when ("N") do;
        fmt = "8.";
      end;

  ***Strictly speaking, a run-time format is not necessary here, as N values are always
  Expressed as whole numbers, but is coded here in the run-time method for symmetry;

      when ("MEAN") do;
        fmt = compress("8." !! put(LBRESDP+1,3.));
      end;

      when ("MIN", "MAX", "MEDIAN") do;
        fmt = compress("8." !! put(LBRESDP,3.));
      end;

      when ("STDDEV") do;
        fmt = compress("8." !! put(LBRESDP+2,3.));
      end;

      otherwise;
    end;
    if stattxt ne "N" or stattxt eq "N" and resarr(statord) then do;
      restext = putN(resarr(statord),fmt);
    end;
    else restext = "";
    output;
  end;
run;
```

Assuming that the output will be displayed per treatment group, column-wise then the data set requires one final transposition before the final PROC REPORT:

```
proc sort data=statlab;
  by VISIT LBTESTCD STATORD STATTXT;
run;

proc transpose data=statlab out=report prefix=TRT;
  by VISIT LBTESTCD STATORD STATTXT;
  var restext;
  id dummytrt;
  idlabel trtlabel;
run;
```

PhUSE 2007

```
proc report data=report nowindows split='-';
  column visit lbtestcd statord stattxt ("Treatment Groups-" TRT0-TRT3);
  define visit / order order=internal "Visit" width=16;
  define lbtestcd / order order=internal "Parameter" width=20;
  define statord / order order=internal noprint;
  define stattxt / display "Statistic" width=9;
  break after visit / skip;
  break after lbtestcd / skip;
run;
```

which produces the following output:

Visit	Parameter	Statistic	Treatment Groups			
			Placebo (N=XX)	Drug A (N=AA)	Drug B (N=BB)	Drug C (N=CC)
BASELINE	ALBUMIN	N	8	8	8	7
		MEAN	45.1	44.1	44.9	45.1
		STDDEV	3.76	2.42	2.80	2.12
		MIN	38	41	41	42
		MAX	49	48	49	47
	ALKALINE PHOSPHATASE	N	8	8	8	7
		MEAN	74.6	78.5	76.0	76.0
		STDDEV	16.24	29.71	11.39	33.97
		MIN	53	54	62	41
		MAX	99	141	93	129
	CALCIUM	N	8	8	8	7
		MEAN	2.381	2.424	2.374	2.436
		STDDEV	0.1076	0.0831	0.0840	0.0998
		MIN	2.15	2.26	2.27	2.27
		MAX	2.50	2.53	2.51	2.54
	GAMMA-GT	N	8	8	8	7
		MEAN	22.5	20.1	32.4	26.6
		STDDEV	11.30	8.06	14.33	15.50
		MIN	12	11	16	14
		MAX	46	32	51	59
	HAEMATOCRIT	N	8	8	8	7
		MEAN	0.450	0.451	0.450	0.434
		STDDEV	0.0378	0.0242	0.0298	0.0276
		MIN	0.37	0.42	0.41	0.39
		MAX	0.49	0.48	0.50	0.47
	HAEMOGLOBIN	N	8	8	8	7
		MEAN	14.90	15.15	14.96	14.41
		STDDEV	1.413	0.987	1.446	1.090
		MIN	11.9	13.5	13.1	12.7
		MAX	16.5	16.2	17.9	15.9
	LEUKOCYTES	N	8	8	8	7
		MEAN	5.894	5.809	6.494	5.699
		STDDEV	0.8463	1.3503	1.2804	0.9570
		MIN	4.12	3.62	5.24	4.62
		MAX	6.95	7.77	8.73	7.49

DECIMAL ALIGNMENT

Decimal alignment may be achieved by using a picture format (instead of the 8.x format used in this example). If this approach is adopted, then care must be taken with the picture format, ensuring that the round option is used in the value definition.

```
picture dp0_ (round)
  low -< 0 = " 0009" (prefix='-')
  0 - high = " 0009"
;
```

This format would then be incorporated in the code below:

```
when ("MEAN") do;
  fmt = compress("dp" !! put(LBRESDP+1,3.) !! "_.");
end;
```

and produce a more aligned result below:

PhUSE 2007

Visit	Parameter	Statistic	Treatment Groups			
			Placebo (N=XX)	Drug A (N=AA)	Drug B (N=BB)	Drug C (N=CC)
BASELINE	ALBUMIN	N	8	8	8	7
		MEAN	45.1	44.1	44.9	45.1
		STDDEV	3.76	2.42	2.80	2.12
		MIN	38	41	41	42
		MAX	49	48	49	47
	ALKALINE PHOSPHATASE	N	8	8	8	7
		MEAN	74.6	78.5	76.0	76.0
		STDDEV	16.24	29.71	11.39	33.97
		MIN	53	54	62	41
		MAX	99	141	93	129
	CALCIUM	N	8	8	8	7
		MEAN	2.381	2.424	2.374	2.436
		STDDEV	0.1076	0.0831	0.0840	0.0998
		MIN	2.15	2.26	2.27	2.27
		MAX	2.50	2.53	2.51	2.54
	GAMMA-GT	N	8	8	8	7
		MEAN	22.5	20.1	32.4	26.6
		STDDEV	11.30	8.06	14.33	15.50
		MIN	12	11	16	14
		MAX	46	32	51	59
	HAEMATOCRIT	N	8	8	8	7
		MEAN	0.450	0.451	0.450	0.434
		STDDEV	0.0378	0.0242	0.0298	0.0276
		MIN	0.37	0.42	0.41	0.39
		MAX	0.49	0.48	0.50	0.47
	HAEMOGLOBIN	N	8	8	8	7
		MEAN	14.90	15.15	14.96	14.41
		STDDEV	1.413	0.987	1.446	1.090
		MIN	11.9	13.5	13.1	12.7
		MAX	16.5	16.2	17.9	15.9
	LEUKOCYTES	N	8	8	8	7
		MEAN	5.894	5.809	6.494	5.699
		STDDEV	0.8463	1.3503	1.2804	0.9570
		MIN	4.12	3.62	5.24	4.62

An alternative, though perhaps longer-winded approach, is to incorporate the rounding of the data into the transpose data step. This requires adding a retext assignment into each branch of the select statement, as the roundings vary for each statistical variable:

```

do statord = 1 to dim(resarr);
  stattxt = statlist(statord);
  if not missing(resarr(statord)) then select(stattxt);
    when ("N") do;
      fmt = "dp0_.";
      if resarr(statord) then do;
        retext = putN(round(resarr(statord),1),fmt);
      end;
      else retext = "";
    end;
    when ("MEAN") do;
      fmt = compress("dp" !! put(LBRESDP+1,3.) !! "_.");
      retext = putN(round(resarr(statord),10**(-(LBRESDP+1))),fmt);
    end;
    when ("MIN", "MAX", "MEDIAN") do;
      fmt = compress("dp" !! put(LBRESDP,3.) !! "_.");
      retext = putN(round(resarr(statord),10**(-(LBRESDP))),fmt);
    end;
    when ("STDDEV") do;
      fmt = compress("dp" !! put(LBRESDP+2,3.) !! "_.");
      retext = putN(round(resarr(statord),10**(-(LBRESDP+2))),fmt);
    end;
    otherwise;
  end;
  output;
end;

```

PhUSE 2007

Knowledge of raising a number to a negative power allows us to use our existing decimal place metadata in LBRDESP to give us the correct number to use in our round-off. So if we have data collected to 2 decimal places, 10-2 gives us 0.01. This rounding is then performed before the format (our picture format without the round function) is applied.

Either way, the reported results are satisfactory in that the correct precision for each parameter is kept with the data by (ironically) binding the format earlier to the data in an intermediate transposition, rather than in the proc report statement.

OTHER APPLICATIONS OF PUTN

PUTN permits the use of formats to be used in a %SYSFUNC call (the PUT function is not allowed). The following allows a formatted SAS date to be assigned, which may be useful in assigning global report headers or footers, without resorting to a data step.

```
title1 "%sysfunc(left(%qsysfunc(date()), worddate18.))";
```

TAKING IT FURTHER

Much of the code can be parameterised and reused across a number of the findings data domains, as defined in SDTM. Whilst I have limited the examples here to the reporting of numerical summary data, the framework could be expanded to databasing the exact format attributes into a LBRFMT variable, containing a SAS format, rather than just recording the number of decimal places.

Alternatively, precision could be stored as the rounding variable mentioned in the alternative transpose data step above, for instance 0.01 indicates the value has been recorded to the nearest 1/100 of a unit. Flexibility in using this approach could be gained by recording conversion factors applied (particularly when non-SI units have been used), for instance the precision could more credibly be recorded as 2.54 (or parts thereof) when inches may have been used instead of cm. The alternative of recording 2 decimal places as a reflection of the data accuracy would in my view be misleading as the cm and inch scales have varying degrees of calibration.

CONCLUSION

This paper has set out a strategy for allowing vertical data structures and parameter attributes to co-exist within a SAS data set. It has also tried to demonstrate that given the right derived or analysis data structure, a report in 3 stages is feasible (Summary, Transpose, Report).

REFERENCES

Wood, Fred et al. *CDISC Analysis Data Model: Version 2.0*, Clinical Data Interchange Standards Consortium, Inc., August 2006
SAS Language Reference: Concepts
SAS Language Reference: Dictionary
Carpenter, Art. *Carpenter's Complete Guide to the SAS Macro Language, 2nd Ed*, SAS Publishing 2004
Staum, Roger. *SAS Software Formats: Going Beneath the Surface*, SUGI 25
Thimbleby, Harold. 1988. "Delaying Commitment." , *IEEE Software*, May, 78–86.

ACKNOWLEDGMENTS

Many thanks are due to Colin Nice who read my first draft and for his encouragement. Also, my former colleagues at Eisai who tolerated my ideas and humour.

PhUSE 2007

RECOMMENDED READING

Bilenas, Jonas V. *The Power of Proc Format*, SAS Press 2005

Cody, Ron. *SAS Functions by Example*, SAS Publishing 2004

McConnell, Steve. *Code Complete*, Second Edition, Microsoft Press 2004

CONTACT INFORMATION

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

Michael Auld

Michael Auld Consulting

Email: sas@w3mac.com

Brand and product names are trademarks of their respective companies.