ABSTRACT
This paper contains several generally unrelated ideas aimed toward the intermediate level pharmaceutical programmer. Believing that length of discussion does not correlate with usefulness, this paper examines: (1) Standardized MedDRA queries (SMQs), (2) a reminder on the proper use of parentheses and logical (or Boolean) operators such as NOT, (3) karyotype or cytogenetic processing, (4) processing and using ALL dates among all SAS data sets in a directory, (5) compressing spaces out of macro variables, (6) dynamically obtaining and using the program name, (7) processing MS Access data bases, and (8) using "where also."

1. STANDARDIZED MEDDRA QUERIES (SMQ)
Examining adverse events (AEs) of special interest has at times been an ad hoc process. Often someone experienced in clinical diagnosis will group a study's individual AEs into event categories thought to be related to the disease of interest or the experimental treatment. The subjective nature of the process results in differences in categorizations from different individuals.

In order to address this, standardized groupings have been created and published by the organization which produces the Medical Dictionary for Regulatory Activities (MedDRA). These standardized groupings are known as Standardized MedDRA Queries (SMQs). As of January 2008, MedDRA has published over 50 SMQs with more scheduled for release in March 2008.

What I give in this paper is a simplification. The official web site for MedDRA containing further information on SMQs is found in the references section below and should be consulted for important details. Also, the website refers to a publication from the Council for International Organizations of Medical Sciences (CIOMS) entitled Development and Rational Use of Standardised MedDRA Queries (SMQs). There are no known regulatory requirements for the use of SMQs. Membership in MedDRA is required to access much of the site and to use SMQs.

In the simplest sense, an SMQ is basically a collection of preferred terms (PTs). Someone experienced in clinical diagnosis will select an SMQ on the basis of its definition and the PTs of which it is comprised. For example, SMQ_List.asc below shows the beginning of the "Torsade de pointes/QT prolongation" definition. The SMQ number associated with "Torsade" is 20,000,001 (SMQs begin with 2 and PTs begin with 1). Looking at SMQ_Content.asc one can see various PTs associated with Torsade, the first being 10,003,109. Printout 1.1 near the end of this section shows that 122 records comprise the Torsade category. If the definition found in SMQ_List.asc is not sufficiently clear, the actual PTs could be examined to clarify the definition of a given SMQ.

Our clinician friend may select several SMQs to reflect a certain medical condition or area of interest. For example, to accurately capture "Cardiac Arrhythmia" the clinician might select SMQs 20,000,050, 20,000,051, 20,000,053, 20,000,055, & 20,000,056. Printout 1.1 again shows the number of records associated with each of these SMQs. Not each of these records is a PT, though. As highlighted in SMQ_Content.asc below, one SMQ may point to another in order to avoid duplication (note that 20,000,050 points to 20,000,053 and that 20,000,053 in turn points to 20,000,055 and 20,000,056). [So in reality, our clinician friend need only point to SMQs 20,000,050 and 20,000,051 since others are encompassed in this selection] Programming can account for this if selections are not completely specified.

Printout 1.2 demonstrates another aspect of working with SMQs: overlap. When multiple SMQs are selected, the possibility exists of having the same PT be a member of more than one SMQ. For example, the first PT (term_code) shown is 10,052,464 and SMQs 20,000,001, 20,000,051, 20,000,056 all contain this PT. This can result in a many-to-many merge issue when joining the SMQ PTs with the AE data set. One way to resolve this is to use an SQL Cartesian join, but this will alter the number of records in the resulting data set. Another way is to create flags showing whether a given PT is or is not a member of the selected grouping. Of course the PT in question may not exist in the AE data set, but this must be checked.

SAS code is provided below to read SMQ_Content.asc and produce the outputs shown below.

/*
 ascii files from MedDRA 10.0 CD. *

SMQ_List.asc
  20000001$Torsade de pointes/QT prolongation (SMQ)$1$Torsade de pointes (TdP) is ...
  ...

SMQ_Content.asc
  20000001$10003109$5$1$A$0$A$7.1$7.1$
  20000001$10003131$5$1$A$0$A$7.1$7.1$
data smq_content;
INFILE 'G:\smq_content.asc' delimiter='$';
input
smq_code
term_code
term_level
term_scope $
term_category $
term_weight $
term_status $
term_addition_version $
term_last_modified_version $
;
run;

data smq_content(drop=term_level);
set smq_content(keep=smq_code term_code term_level
where=(term_level NOT IN (0) & smq_code IN (20000050 20000051 20000053 20000055 20000056 20000001)));
select (smq_code);
when (20000050) SMQ_DESC='Cardiac arrhythmia';
when (20000051) SMQ_DESC='Cardiac arrhythmia';
when (20000053) SMQ_DESC='Cardiac arrhythmia';
when (20000055) SMQ_DESC='Cardiac arrhythmia';
when (20000056) SMQ_DESC='Cardiac arrhythmia';
when (20000001) SMQ_DESC='Torsade de Pointes';
*otherwise;
end;
label SMQ_DESC = 'Pharmion description of SMQ code'
SMQ_CODE = 'SMQ Code from smq_content.asc v10.0'
term_code = 'PT / LLT from smq_content.asc v10.0';
run;

proc freq noprint;
tables SMQ_DESC*smq_code / missing list out=qwe(drop=percent);
run;

title '1.1 Frequency of PTs in each given SMQ.';
proc print width=min data=qwe;
by smq_desc;
id smq_desc;
run;

proc sort data=smq_content;
by term_code smq_code;
run;

data smq_content2;
set smq_content;
by term_code smq_code;
if NOT (first.term_code EQ 1 & last.term_code EQ 1) ;
run;

title '1.2 Selected PTs showing overlapping SMQs.';
proc print width=min
   data=smq_content2(where=(term_code IN (10052464 10052509 10052810)));
   by term_code;
   id term_code;
   run;
endsas;

1.1 Frequency of PTs in each given SMQ.

<table>
<thead>
<tr>
<th>SMQ_DESC</th>
<th>SMQ_CODE</th>
<th>COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cardiac arrhythmia</td>
<td>20000050</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>20000051</td>
<td>140</td>
</tr>
<tr>
<td></td>
<td>20000053</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>20000055</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>20000056</td>
<td>148</td>
</tr>
<tr>
<td>Torsade de Pointes</td>
<td>20000001</td>
<td>122</td>
</tr>
</tbody>
</table>

1.2 Selected PTs showing overlapping SMQs.

<table>
<thead>
<tr>
<th>TERM_CODE</th>
<th>SMQ_CODE</th>
<th>SMQ_DESC</th>
</tr>
</thead>
<tbody>
<tr>
<td>10052464</td>
<td>20000001</td>
<td>Torsade de Pointes</td>
</tr>
<tr>
<td></td>
<td>20000051</td>
<td>Cardiac arrhythmia</td>
</tr>
<tr>
<td></td>
<td>20000056</td>
<td>Cardiac arrhythmia</td>
</tr>
<tr>
<td>10052509</td>
<td>20000001</td>
<td>Torsade de Pointes</td>
</tr>
<tr>
<td></td>
<td>20000051</td>
<td>Cardiac arrhythmia</td>
</tr>
<tr>
<td></td>
<td>20000056</td>
<td>Cardiac arrhythmia</td>
</tr>
<tr>
<td>10052810</td>
<td>20000001</td>
<td>Torsade de Pointes</td>
</tr>
<tr>
<td></td>
<td>20000051</td>
<td>Cardiac arrhythmia</td>
</tr>
</tbody>
</table>

2. USING PARENTHESES LOGICALLY
The example below is a reminder to pay attention to the proper use of parentheses and logical (or Boolean) operators such as NOT. First, the following two sentences from the SAS manual are important to understand.

(1) In SAS, any numeric value other than 0 or missing is true, and a value of 0 or missing is false.

(2) The result of NOT in front of a quantity with a nonzero, nonmissing value is 0 (false).

Now, on to discuss the flags:

Flag1 is straightforward. When the value of x is not equal to 1, flag1 is populated with a 1.

Flag2 is similarly straightforward. The expression inside the parentheses is evaluated first. Flag2 is populated as 1 only when the expression inside the parentheses evaluates to false since the NOT operator reverses this false to make the entire statement true.

Flag3 may be a bit perplexing. One might be tempted to evaluate the x=1 expression first and then apply the NOT, but the expression must be evaluated left to right. First, sentence (1) above is applied in evaluating the logical value of x. This means that the missing and zero observations of x have a logical value of 0 (false) and the 1, 2, & 3 values of x have a logical value of 1 (true). Then (2) is applied, meaning the NOT negates (takes the opposite of) the logical value. Finally, the =1 part is evaluated which simply maintains the existing logical value, just like multiplying any number by 1 results in the original number.

Flag4 is simply a more explicit version of Flag3, demonstrating that the "NOT x" part of the expression is evaluated first.

Flag5 is an application of sentence (1) above.

Flag6 is also an application of sentence (1) above.
data any;
  input x @@;
  if x NE 1 then flag1=1;
  if NOT (x=1) then flag2=1;
  if NOT x=1 then flag3=1;
  if (NOT x)=1 then flag4=1;
  if NOT x then flag5=1;
  if x then flag6=1;
cards;
  . 0 1 2 3
;
runc;
proc print width=min noobs;
rund;
/* * OUTPUT * * * * * * * * * * * * * * * * * * * * * * * * * * */

<table>
<thead>
<tr>
<th>x</th>
<th>flag1</th>
<th>flag2</th>
<th>flag3</th>
<th>flag4</th>
<th>flag5</th>
<th>flag6</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>.</td>
</tr>
<tr>
<td>1</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>1</td>
</tr>
</tbody>
</table>

3. KARYOTYPE OR CYTOGENETIC PROCESSING

The rationale for including a section dealing with cell biology in this paper is simply to share what I've used and my limited knowledge in the hope that other programmers will benefit and extend our common knowledge in this important area. I hope to first provide some context and then to suggest an approach to working with human chromosome nomenclature. I am not an expert in this area.

Chromosomes are located in the nuclei of most human cells, and most human cells contain 23 different pairs of chromosomes. Illustrations of human chromosomes were first published over 125 years ago. Soon after, the term cytogenetics was used to refer to the study of chromosomes, combining cytology (cell biology) and genetics. A karyotype is the ordered arrangement of chromosomes and karyotyping is the description of chromosomal abnormalities. Over the past 50 years, efforts to standardize the description of the human chromosome and its abnormalities have been driven by the International System for Human Cytogenetic Nomenclature. The culmination of their latest conference is published in ISCN (2005).

Advances in technology and increased knowledge about how abnormalities in the human chromosome are related to physical impairment including disease have accompanied this standardization in karyotyping. Cytogenetic analysis has become an important risk factor for diagnosing and predicting disease progression and for categorizing stages of various diseases, particularly cancer. Along with this progress, karyotype descriptions are being collected in clinical trials and programmers are being asked to process this information. What follows is an overview of karyotype descriptions - just enough to give a flavor but certainly not intended to substitute for further study. The examples and descriptions are taken from Gersen (1999).

The normal human karyotypes for a female and male, respectively, are 46,XX and 46,XY. The number before the comma is the number of chromosomes found in the cell and the letters after the comma represent the sex of the individual, with maleness represented by the presence or absence of the Y chromosome. In a normal human cell, there are 22 "autosome" (non-sex) pairs numbered as pairs 1 through 22 accounting for 44 of the 46 chromosomes plus two sex chromosomes accounting for the remaining two chromosomes, with these two represented as XX or XY. The pairs are assigned numbers according to length, size and centromere position (where the pairs join) and are unambiguously distinguishable from each other.

Abnormalities in the number of autosome or sex chromosomes are represented as deviations from this normal karyotype. For example, 45,X represents a female with only one sex chromosome in what is called Classical monosomy X or Turner syndrome (mono indicates one). 47,XXY represents a male with an extra X chromosome in what is called classical Klinefelter syndrome.

As seen, the description of a karyotype begins with the total number of chromosomes, including sex chromosomes, followed by a comma, followed by details on the sex chromosomes. If an abnormality in any of the autosome pairs is present, a description of this abnormality is given, each preceded by a comma. All autosome abnormalities are ordered by number from smallest number to largest. For example, 48,XX,+18,+21 indicates a female with an extra 18th and an extra 21st chromosome, or trisomy 18 and trisomy 21 (tri indicates three).

Each chromosome has a centromere which divides the short or p arm from the long or q arm. The light and dark bands along the arms comprise the regions of the arm. So, the last (7th, in this case) band of the 3rd region on the long arm of
chromosome 2 is referenced as 2q37. If a part of an arm is missing, the abbreviation \textit{del} (for deletion) is used. So \textit{del}(1) (p21p32) indicates that the short arm segment of chromosome 1 between region 2 band 1 and region 3 band 2 has been deleted. Many abnormality abbreviations exist such as deletions, additions, translocations, duplications, insertions, and inversions. These are documented in chapter 3 of ISCN (2005).

Generally, more than one cell is karyotyped. If not all cells follow the same pattern, the cells are grouped into like "clones" and the number of cells in each clone is designated within square brackets with the separate clones distinguished with a slash between them. Generally spaces are not allowed in the ISCN standard except in certain cases such as between an abbreviation and a number. So returning one last time to our standard nomenclature, we have seen the meaning and can interpret many of the parts of the following karyotype (from Gersen (1999), page 56):

\begin{verbatim}
47,XY,del(5)(q13q33),+8,t(9;22)(q34;q11.2)[4]/48,idem,+9,i(17)(q10)[12]/46,XY[4]
\end{verbatim}

We first recognize that the slashes show three different clone lines of 20 (4+12+4, the numbers in the square brackets) cells:

\begin{verbatim}
47,XY,del(5)(q13q33),+8,t(9;22)(q34;q11.2)[4]
48,idem,+9,i(17)(q10)[12]
46,XY[4]
\end{verbatim}

There are four cells in the first clone line and each cell has three abnormalities. First, there is a deletion in chromosome 5. Second, there is an additional chromosome 8. It is this second abnormality that causes the number of chromosomes shown at the beginning of the karyotype description to be 47 rather than 46. Third, there is a translocation in the long arms of chromosomes 9 and 22.

There are 12 cells in the second clone line and five abnormalities. The first three abnormalities are identical to those seen in the first clone line. This is represented with "idem." The fourth abnormality is an additional chromosome 9. This adds one to the 47 at the beginning of the first clone line to make 48. The fifth abnormality is a problem on the long arm of chromosome 17.

There are four cells in the third clone line and these cells are all normal.

So as a programmer, we see a well-defined nomenclature describing an important piece of clinical research. To say that there is much more to it than what I have shown would be a gross understatement, but by learning a few key parts of the nomenclature we can add value to the process by parsing a karyotype into useful information. For example, understanding the slash, the comma, the square bracket, and a few common abbreviations and symbols will go a long way toward this goal.

Programmers with even this limited ability can contribute in at least two ways. First, we can learn enough to identify the presence or absence of a particular cytogenetic abnormality. In some studies an expert reviewer populates check boxes with this information, and in such cases we can process the karyotype description independently to confirm or contest this characterization. Second, we can identify errors in data entry based on knowing the standard nomenclature. Of course some of these may be actual transcription errors – it is likely that the error rate for such cryptic information is higher than other types of clinical trial data. Other errors may go back to the expert reviewers themselves.

4. IDENTIFYING A PATIENT'S LAST DATE AMONG ALL DATA SETS IN A DIRECTORY
Let's say your task is to populate a 'last known date of contact' for a patient. The following example shows how this can be done dynamically and shows some useful by-products along the way.

Let's assume you have a directory containing 30 data sets which contain zero, one, or more date variables. The following code dynamically searches through the formats of all variables in all 10 data sets in order to identify the variables with a date format. It then cycles through these variables, setting together records containing patient, data set, variable name, and date. Printout 4.1 below shows that among all 30 data sets, date records were found in five variables across three files (AE.STARTDT, AE.ENDDT, CM.STARTDT, CM.ENDDT, & DM.DOB). From this we use last dot processing and a transpose to end up with our result, shown in Printout 4.2.

Along the way we can detect both missing and nonsense data. Note in Printout 4.1 we have dates without patient numbers (observations 15-16 & 29-30). Note also that in Printout 4.2 we have a future date (PT=2 has a 12Dec2008 date and this text is being written in \textbf{February} 2008). The same could be true for early out-of-range dates such as dates in the early 1900s for a pediatric trial.

There are a number of details that are of interest.

(1) The macro variables generated with this code are shown below in the log section below the printouts.
(2) We can augment the list of date variables to include more than those qualifying via the date9. format.

(3) We can omit certain variables if they do not contribute to our purpose. Common variables in this class are: Investigator Signature, Date of Central Review, System Variables such as Date Entered. Similarly, dates after patient death should be critically viewed with regard to our purpose. For example, a date of last known contact could be a follow-up call to a family member.

(4) Of course just because you arrive at an answer does not mean it makes any sense. Most if not all conmeds shown in this example have the patient starting the medicine AFTER ending it.

(5) Note that DM.DOB is not represented in Printout 4.2. This makes sense since a date of birth would never reasonably be the last date available for a patient in a clinical study.

(6) In this example, a cutoff date of 19Dec2007 is applied in creating the final variable of Printout 4.2.

```
libname raw 'U:\Conferences\Data\raw';
%
macro adhoc014;
proc sql noprint;
  select memname, name
  into :memname_ separated by ' ',
       :name_ separated by ' ' 
  from dictionary.columns
  where upcase(libname)='RAW' & index(upcase(format),'DATE9') GT 0;
quit;
%put _all_

data all_dates;
  format date date9.;
run;
%
macro loop;
%let i=1;
%let ids=%scan(&memname_,&i,%str( ));
%let ivn=%scan(   &name_,&i,%str( ));
%do %while (%length(&ids) gt 0);
  proc sort nodupkey out=&ids 
data=raw.&ids(where=(date NE .)
   keep=pt &ivn rename=(&ivn=date));
  by pt date;
  run;
  data &ids;
  set &ids;
  dataset="&ids";
  varname="&ivn";
  run;
  data all_dates(where=(date NE .));
  set all_dates &ids;
  run;
%let i=%eval(&i+1);
%let ids=%scan(&memname_,&i,%str( ));
%let ivn=%scan(   &name_,&i,%str( ));
%end;
%mend loop;
%loop;

proc print width=min data=all_dates(obs=333);
title '4.1 Raw data, stacked.';
run;

proc sort data=all_dates;*(where=(date NE .));
  by pt date dataset varname;
run;
data all_dates;
set all_dates;
```
by pt date;
if last.pt;
v=compress(dataset||'_'||varname);
run;
proc sort data=all_dates(drop=dataset varname);
by pt v;
run;
proc sql noprint;
select distinct v
into :v_ separated by ' ' 
from all_dates;
quit;
%put &v_;
***run;

proc transpose data=all_dates out=all_dates_t(drop=_);
by pt;
var date;
id v;
run;
data lastDT;
set all_dates_t;
format lastDT lastDT_cutoff date9.;
lastDT=sum(of &v_);
lastDT_cutoff=min('19DEC2007'd,sum(of &v_));
run;
proc print width=min;
title '4.2  Last date by patient and variable. Cutoff implemented.';
run;
%mend adhoc014;
%adhoc014;

4.1 Raw data, stacked.

<table>
<thead>
<tr>
<th>Obs</th>
<th>date</th>
<th>PT</th>
<th>dataset</th>
<th>varname</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>07DEC1941</td>
<td>1</td>
<td>AE</td>
<td>STARTDT</td>
</tr>
<tr>
<td>2</td>
<td>31DEC1999</td>
<td>1</td>
<td>AE</td>
<td>STARTDT</td>
</tr>
<tr>
<td>3</td>
<td>12DEC2008</td>
<td>2</td>
<td>AE</td>
<td>STARTDT</td>
</tr>
<tr>
<td>4</td>
<td>01JAN2000</td>
<td>3</td>
<td>AE</td>
<td>STARTDT</td>
</tr>
<tr>
<td>5</td>
<td>16JAN1991</td>
<td>4</td>
<td>AE</td>
<td>STARTDT</td>
</tr>
<tr>
<td>6</td>
<td>11SEP2001</td>
<td>4</td>
<td>AE</td>
<td>STARTDT</td>
</tr>
<tr>
<td>7</td>
<td>01JUL1997</td>
<td>7</td>
<td>AE</td>
<td>STARTDT</td>
</tr>
<tr>
<td>8</td>
<td>08MAY1945</td>
<td>1</td>
<td>AE</td>
<td>ENDDT</td>
</tr>
<tr>
<td>9</td>
<td>09NOV1989</td>
<td>1</td>
<td>AE</td>
<td>ENDDT</td>
</tr>
<tr>
<td>10</td>
<td>15DEC2007</td>
<td>2</td>
<td>AE</td>
<td>ENDDT</td>
</tr>
<tr>
<td>11</td>
<td>14AUG1945</td>
<td>3</td>
<td>AE</td>
<td>ENDDT</td>
</tr>
<tr>
<td>12</td>
<td>26MAR1990</td>
<td>4</td>
<td>AE</td>
<td>ENDDT</td>
</tr>
<tr>
<td>13</td>
<td>09NOV1992</td>
<td>4</td>
<td>AE</td>
<td>ENDDT</td>
</tr>
<tr>
<td>14</td>
<td>08MAY2000</td>
<td>7</td>
<td>AE</td>
<td>ENDDT</td>
</tr>
<tr>
<td>15</td>
<td>11AUG2000</td>
<td>.</td>
<td>CM</td>
<td>STARTDT</td>
</tr>
<tr>
<td>16</td>
<td>30JUN2004</td>
<td>.</td>
<td>CM</td>
<td>STARTDT</td>
</tr>
<tr>
<td>17</td>
<td>23AUG2000</td>
<td>2</td>
<td>CM</td>
<td>STARTDT</td>
</tr>
<tr>
<td>18</td>
<td>25AUG2000</td>
<td>2</td>
<td>CM</td>
<td>STARTDT</td>
</tr>
<tr>
<td>19</td>
<td>30AUG2000</td>
<td>2</td>
<td>CM</td>
<td>STARTDT</td>
</tr>
<tr>
<td>20</td>
<td>01SEP2000</td>
<td>3</td>
<td>CM</td>
<td>STARTDT</td>
</tr>
<tr>
<td>21</td>
<td>10MAY2000</td>
<td>4</td>
<td>CM</td>
<td>STARTDT</td>
</tr>
<tr>
<td>22</td>
<td>23JUL2003</td>
<td>5</td>
<td>CM</td>
<td>STARTDT</td>
</tr>
<tr>
<td>23</td>
<td>22OCT2003</td>
<td>5</td>
<td>CM</td>
<td>STARTDT</td>
</tr>
<tr>
<td>24</td>
<td>28APR2004</td>
<td>5</td>
<td>CM</td>
<td>STARTDT</td>
</tr>
<tr>
<td>25</td>
<td>22AUG2000</td>
<td>6</td>
<td>CM</td>
<td>STARTDT</td>
</tr>
<tr>
<td>26</td>
<td>14NOV2000</td>
<td>6</td>
<td>CM</td>
<td>STARTDT</td>
</tr>
<tr>
<td>27</td>
<td>19MAR2001</td>
<td>6</td>
<td>CM</td>
<td>STARTDT</td>
</tr>
<tr>
<td>28</td>
<td>08MAY2001</td>
<td>6</td>
<td>CM</td>
<td>STARTDT</td>
</tr>
</tbody>
</table>
4.2 Last date by patient and variable. Cutoff implemented.

<table>
<thead>
<tr>
<th>Obs</th>
<th>PT</th>
<th>CM_STARTDT</th>
<th>AE_STARTDT</th>
<th>AE_ENDDT</th>
<th>lastDT</th>
<th>cutoff</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>.</td>
<td>30JUN2004</td>
<td>.</td>
<td>.</td>
<td>30JUN2004</td>
<td>30JUN2004</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>.</td>
<td>31DEC1999</td>
<td>.</td>
<td>31DEC1999</td>
<td>31DEC1999</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>01SEP2000</td>
<td>.</td>
<td>.</td>
<td>01SEP2000</td>
<td>01SEP2000</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>28APR2004</td>
<td>.</td>
<td>.</td>
<td>28APR2004</td>
<td>28APR2004</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>08MAY2001</td>
<td>.</td>
<td>.</td>
<td>08MAY2001</td>
<td>08MAY2001</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>.</td>
<td>08MAY2000</td>
<td>.</td>
<td>08MAY2000</td>
<td>08MAY2000</td>
</tr>
</tbody>
</table>

MLOGIC(ADHOC014): %PUT _all_
ADHOC014 NAME_ STARTDT ENDDT STARTDT ENDDT DOB
ADHOC014 MEMNAME_ AE AE CM CM DM

MLOGIC(ADHOC014): %PUT &V_
SYMBOLGEN: Macro variable V_ resolves to AE_ENDDT AE_STARTDT CM_STARTDT
AE_ENDDT AE_STARTDT CM_STARTDT
5. COMpressing Spaces Out of Macro Variables

The "separated by ' ' " SQL phrase is a quick way to ensure no leading or trailing spaces exist in a macro variable. This alleviates the need for post-processing the macro variable with a "call symput('e1a',compress('&e1a'));" statement. The following code counts the number of unique patients (PT) in data set DS1 in treatment group 1 and places the result in a macro variable named E1A which has no leading or trailing spaces. Of course the design of this SEPARATED BY phrase is to specify the delimiter between the individual parts of a compound macro variable; the fact that leading and trailing spaces do not exist when the macro variable contains only a single item is a useful by-product.

```sas
proc sql noprint;
    select count(distinct pt) into: E1A separated by ' ' from DS1 where trtgrp=1;
quit;
```

6. Obtaining and Using the Program Name

Several versions ago SAS made available the automatic macro variable SYSPROCESSNAME. This stores the current program name and is available for use just like any other local, global, or automatic macro variable. The example below shows how to extract a useful part of this macro variable and use it in a title statement. Many other automatic macro variables are available and can be viewed using the %put _all_; statement as shown.

```sas
options nonumber nodate nocenter;
titiel %SUBSTR(&SYSPROCESSNAME,9) "  &sysdate9 &systime";
data any;
x=1;
run;
proc print width=min noobs;
run;
%put _all_;
%put SYSPROCESSNAME=&SYSPROCESSNAME;
```

7. Processing a Microsoft Access Data Base

Let’s say you have a CD containing a Microsoft Access data base that you want to read with SAS (assume you do not have MS Access). You can see the data base name via, say, MS Explorer, but you do not know the individual file (table) names within the data base.

This can be accomplished by using the code below. You need to identify the Access file as shown in the libname statement, then use this libname in the CONTENTS procedure as shown, along with the wildcard _all_. Part 1 of the output shows all tables in the data base, in this case 8. Part 2 gives the standard CONTENTS output of each of the 8 individual tables. By noticing the way the individual file is referenced in the Part 2 PROC CONTENTS output (MYMDB.'NR Names for Mailing'n) one can replicate this in manipulating the data set further. This is shown in the PRINT procedure, though the PRINT output has been omitted.

```sas
libname mymdb "D:\Names for Mailing.mdb";
proc contents data=mymdb._all_; run;
proc print width=min data=MYMDB.'NR Names for Mailing'n(obs=4); run;
```
(PART 1) The CONTENTS Procedure

Directory
Libref MYMDB
Engine ACCESS
Physical Name D:\Names for Mailing.mdb
Schema/Owner Admin

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Type</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NR Names for Mailing</td>
<td>DATA</td>
<td>TABLE</td>
</tr>
<tr>
<td>2</td>
<td>NR Names for Mailing 2</td>
<td>DATA</td>
<td>TABLE</td>
</tr>
<tr>
<td>3</td>
<td>NR hunt combo</td>
<td>DATA</td>
<td>TABLE</td>
</tr>
<tr>
<td>4</td>
<td>NR hunt combo 2</td>
<td>DATA</td>
<td>TABLE</td>
</tr>
<tr>
<td>5</td>
<td>Res Names Mailing</td>
<td>DATA</td>
<td>TABLE</td>
</tr>
<tr>
<td>6</td>
<td>Res Names Mailing 2</td>
<td>DATA</td>
<td>TABLE</td>
</tr>
<tr>
<td>7</td>
<td>Resident hunt combo</td>
<td>DATA</td>
<td>TABLE</td>
</tr>
<tr>
<td>8</td>
<td>Resident hunt combo 2</td>
<td>DATA</td>
<td>TABLE</td>
</tr>
</tbody>
</table>

(Part 2) The CONTENTS Procedure

Data Set Name MYMDB.'NR Names for Mailing'n Observations .
Member Type DATA Variables 12
Engine ACCESS Indexes 0

Alphabetic List of Variables and Attributes

<table>
<thead>
<tr>
<th>#</th>
<th>Variable</th>
<th>Type</th>
<th>Len</th>
<th>Format</th>
<th>Informat</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>CUST_ADTL_ADDRESS</td>
<td>Char</td>
<td>40</td>
<td>$40.</td>
<td>$40.</td>
<td>CUST_ADTL_ADDRESS</td>
</tr>
<tr>
<td>7</td>
<td>CUST_CITY</td>
<td>Char</td>
<td>40</td>
<td>$40.</td>
<td>$40.</td>
<td>CUST_CITY</td>
</tr>
<tr>
<td>2</td>
<td>CUST_ID</td>
<td>Num</td>
<td>8</td>
<td>10.</td>
<td>10.</td>
<td>CUST_ID</td>
</tr>
<tr>
<td>4</td>
<td>CUST_NAME</td>
<td>Char</td>
<td>40</td>
<td>$40.</td>
<td>$40.</td>
<td>CUST_NAME</td>
</tr>
</tbody>
</table>

8. AUGMENTING WHERE STATEMENTS IN PROCEDURES

The example below demonstrates that multiple WHEREs can be used but can result in output different than one might expect.
First, note that a WHERE= can be used directly as a data set option and a WHERE can be used as a statement. Second, note that this discussion deals only with procedures; SAS documentation is clear regarding what happens in the DATA STEP: "If you use both the WHERE= data set option and the WHERE statement in the same DATA step, SAS ignores the WHERE statement for data sets with the WHERE= data set option."

In 8.1 we see the full data set printed since no WHEREs have been specified.

In 8.2 (and all subsequent examples) we see the WHERE= data set option applied resulting in the a=5 observation being omitted.

In 8.3, in addition to the WHERE=, we have one WHERE statement active and this statement augments the WHERE= since a=4 is omitted.

In 8.4 we see that a=1 has been omitted due to the final WHERE statement but that a=4 has returned. This demonstrates that only the last WHERE statement is used: previous WHERE statements are ignored.

In 8.5 we see that both WHERE statements are in effect since an ALSO is used in the second WHERE statement.

It should be noted that the statement "Where clause has been augmented" found in the log could be misconstrued to mean that the WHERE statements are somehow cumulative. While the WHERE= data set option is supplemented (not superseded) by WHERE statements, only the last WHERE statement is used, and subsequent WHERE ALSO statements add to the subset condition. A nice treatment of WHERE and subsetting IF statements can be seen in Gupta (2007).
data any;
  input a @@;
cards;
  1 2 3 4 5
;
run;
%macro loop(s=,w1=,w2=,w3=);
title1 "&s  w1=&w1, w2=&w2, w3=&w3"
proc print width=min data=any(&w1) noobs;
  &w2;
  &w3;
run;
%mend loop;
%loop(s=8.1, w1=                   ,w2=                 ,w3=                 );
%loop(s=8.2, w1=where=(a LE 4)     ,w2=                 ,w3=                 );
%loop(s=8.3, w1=where=(a LE 4)     ,w2=where a LE 3     ,w3=                 );
%loop(s=8.4, w1=where=(a LE 4)     ,w2=where a LE 3     ,w3=where      a GE 2);%loop(s=8.5, w1=where=(a LE 4)     ,w2=where a LE 3     ,w3=where ALSO a GE 2);run;
8.1  w1=, w2=, w3=
  a
  1
  2
  3
  4

8.2  w1=where=(a LE 4), w2=, w3=
  a
  1
  2
  3

8.3  w1=where=(a LE 4), w2=where a LE 3, w3=
  a
  1
  2
  3

8.4  w1=where=(a LE 4), w2=where a LE 3, w3=where      a GE 2
  a
  2
  3
  4

8.5  w1=where=(a LE 4), w2=where a LE 3, w3=where ALSO a GE 2
  a
  2
  3

It should also be noted that macro parameters can be set up to augment a single where statement or data step option. For example, %let andwhere=& a GE 2; defines the macro variable andwhere which can be used this way.
REFERENCES


ISCN (2005): An International System for Human Cytogenetic Nomenclature, Shaffer L.G., Tommerup N. (eds); S. Karger, Basel, Switzerland, 2005

Section 2 above references: SAS Operators in Expressions
Section 8 above references: WHERE= Data Set Option

ACKNOWLEDGMENTS

My thanks goes to the many who have freely shared ideas over the years. Special thanks goes to Lela Lucy who reviewed the SMQ section of this paper, Dr. Indra Fernando who provided useful expertise, and Julie Greene who provided administrative assistance.

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. © indicates USA registration.

CONTACT INFORMATION

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

John Morrill
Pharmion Corporation
9900 W 109th Street, Suite 300
Overland Park, KS  66210-1438
Work Phone: 913-266-0300
E-mail: JMorrill@Pharmion.com or JSMorrill@gmail.com