PharmaSUG 2017 – Paper BB02

Making Documents 'Intelligent' with Embedded Macro Calls, DOSUBL and Proc STREAM: An example with the CONSORT Flow Diagram

Joseph Hinson, inVentiv Health, Princeton, NJ, USA

ABSTRACT

Documents can be considered 'intelligent' if they can self-process parts of themselves. One way is to embed them with macro elements. Such macro-laden documents can then be placed on the SAS® Input Stack for macro processing. However documents like RTF, XML, and HTML, tend to have extraneous codes that would violate SAS syntax if placed on the Input Stack as is. Placing non-SAS documents on the Input Stack therefore requires the STREAM Procedure, which by disabling the SAS Compiler can allow the intact document codes mixed with macro elements. Once the Macro Facility has resolved all the macro elements, the document is streamed back to a file location. Having document templates embedded with macro variables is nothing new, but until now, the role had been limited to just text substitutions. By embedding documents with actual macro calls, self-processing of documents becomes possible. In such cases, the DATA steps and procedures within the macros need to be wrapped inside a DOSUBL function and called with a %SYSFUNC, to force computation and prevent SAS code from being streamed out to the output file. Such an approach is ideal for documents like the Consolidated Standards Of Reporting Trials (CONSORT) Flow Diagram, which depicts the progress through the phases of a clinical trial (enrollment, intervention allocation, follow-up, and data analysis) by showing the counts of study participants for each phase. With an intelligent CONSORT template, the counts are replaced by macro calls such that the flow diagram undergoes self-processing when passed through Proc STREAM.

INTRODUCTION

Companies have long known that many business documents have common unchanging components. Thus a regular time-saving feature is creating templates for producing a variety of documents. Such templates typically have "boiler-plate" sections -- the parts that never change from document to document. The templates also may have place-holders for the document-specific text (sometimes called "tokens"). Typical tokens include "::ClientCompanyName::","::ClientCompanyAddress::", "::ClientPrimaryContactFirstName::", etc. With SAS, such templates can have macro variables as "tokens". A number of commercial products (eg *PandaDoc, Nitro*) have emerged based on that template principle. But all these template approaches have just the same principle: text-substitution for what changes. To the best of the author's knowledge, there is no such thing as a document template with embedded software elements that can do actual computations.

Macro calls within documents is the feature that can make documents appear smart. Consider for example, a clinical document template that can be updated with different patients to generate patient-specific reports. If this template is "smart", it can figure out the gender of the patient and use the appropriate "he" or "she", "his" or "her". If the patient is a male, or an 80-year old female, or an infant, the smart document wouldn't bother displaying "Pregnancy Test Results", just as "Prostate Specific Antigen Test Results" would be skipped for a female patient. Similarly, the document can assess the patient's country and use the appropriate units of measurements, type of currency, and whether surnames come before given names. An intelligent document can figure out the correct spelling of "hematology" as opposed to "haematology", based on the country in question. Macros embedded in intelligent documents can retrieve information and process such choices programmatically.

With two new SAS tools in SAS 9.4 – DOSUBL function and Proc STREAM-- documents can now do actual computations and populate themselves with the numerical or textual results. Proc STREAM allows RTF documents to be placed on the SAS Input Stack for macro processing, without any syntax errors triggered by the RTF codes. The DOSUBL function forces macros to execute their DATA step and PROC codes before being streamed back onto the Input Stack. RTF documents therefore can contain macro calls for macros that process DATA and PROC steps. This is the basis for creating intelligent documents with SAS.

PROC STREAM

The SAS 9.4 Proc STREAM was designed for processing documents containing macro elements and streaming the result to an external file. But this general idea of adding macro elements to documents for macro resolution would only be fairly easy if one could engage just the macro processor and not the SAS Compiler. For as such a document is tokenized, macro elements would be diverted to the macro processor and the resolved values would be returned and the document streamed back in original form except with resolved macro values. This is accomplished with Proc

STREAM, with which almost anything can be put on the Input Stack (with the exception of binary formats like PDF, JPEG, DOCX). Proc STREAM disables the SAS Compiler thereby permitting text containing SAS syntax-violating RTF codes. After macro resolutions, the RTF document is streamed to an external file location with the original structure and non-macro contents intact. Without Proc STREAM, the log would display syntax errors from the RTF codes, as shown below for a simple RTF table:

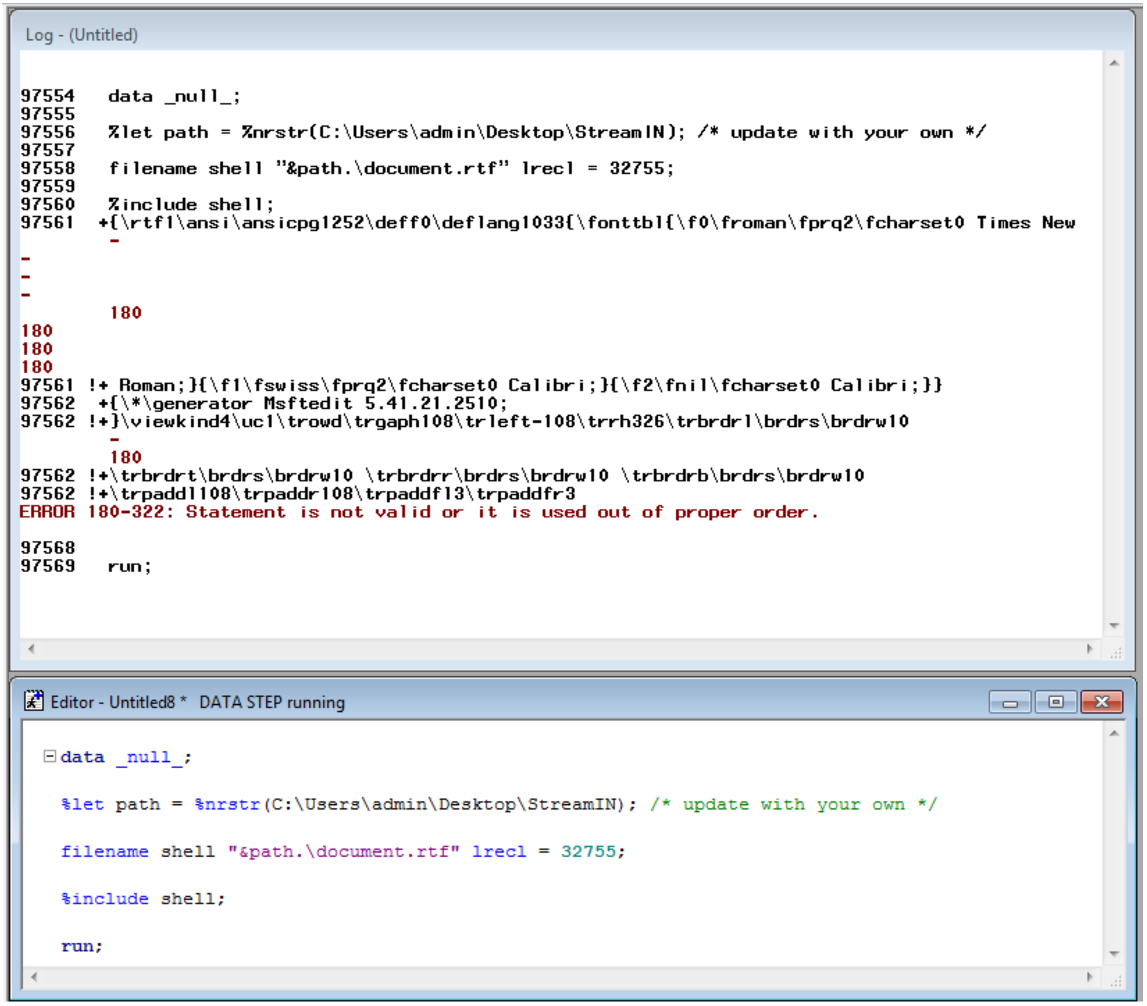
(a) RTF Table (document.rtf):

ACTIVE	PLACEBO
x.xx	x.xx
xx.x	xx.x

(b) Underlying RTF Code:

```
{\rtf1\ansi\ansicpg1252\deff0\deflang1033{\fonttbl{\f0\froman\fprq2\fcharset0 Times New Roman;}{\f1\fswiss\fprq2\fcharset0
Calibri;}{\f2\fnil\fcharset0 Calibri;}}

{\*\generator Msftedit 5.41.21.2510;}\viewkind4\uc1\trowd\trgaph108\trleft-108\trrh326\trbrdrl\brdrs\brdrw10 \trbrdrt\brdrs\brdrw10
\trbrdrr\brdrs\brdrw10 \trbrdrb\brdrs\brdrw10 \trpaddl108\trpaddr108\trpaddfl3\trpaddfr3

\clbrdrl\brdrw10\brdrs\clbrdrt\brdrw10\brdrs\clbrdrr\brdrw10\brdrs\clbrdrb\brdrw10\brdrs
\cellx1503\clbrdrl\brdrw10\brdrs\clbrdrt\brdrw10\brdrs\clbrdrr\brdrw10\brdrs\clbrdrb\brdrw10\brdrs \cellx3114\pard\intbl\f1\fs22
ACTIVE\cell PLACEBO\cell\row\trowd\trgaph108\trleft-108\trrh326\trbrdrl\brdrs\brdrw10 \trbrdrt\brdrs\brdrw10 \trbrdrr\brdrs\brdrw10
\trbrdrb\brdrs\brdrw10 \trpaddl108\trpaddr108\trpaddfl3\trpaddfr3

\clbrdrl\brdrw10\brdrs\clbrdrt\brdrw10\brdrs\clbrdrr\brdrw10\brdrs\clbrdrb\brdrw10\brdrs
\cellx1503\clbrdrl\brdrw10\brdrs\clbrdrt\brdrw10\brdrs\clbrdrr\brdrw10\brdrs\clbrdrb\brdrw10\brdrs \cellx3114\pard\intbl x.xx\cell
x.xx\cell\row\trowd\trgaph108\trleft-108\trrh326\trbrdrl\brdrs\brdrw10 \trbrdrt\brdrs\brdrw10 \trbrdrr\brdrs\brdrw10
\trbrdrb\brdrs\brdrw10 \trpaddl108\trpaddr108\trpaddfl3\trpaddfr3

\clbrdrl\brdrw10\brdrs\clbrdrt\brdrw10\brdrs\clbrdrr\brdrw10\brdrs\clbrdrb\brdrw10\brdrs
\cellx1503\clbrdrl\brdrw10\brdrs\clbrdrt\brdrw10\brdrs\clbrdrr\brdrw10\brdrs\clbrdrb\brdrw10\brdrs \cellx3114\pard\intbl xx.x\cell
xx.x\cell\row\pard\sa200\sl276\slmult1\lang9\f2\par

}
```

(a) SAS Code and Log:



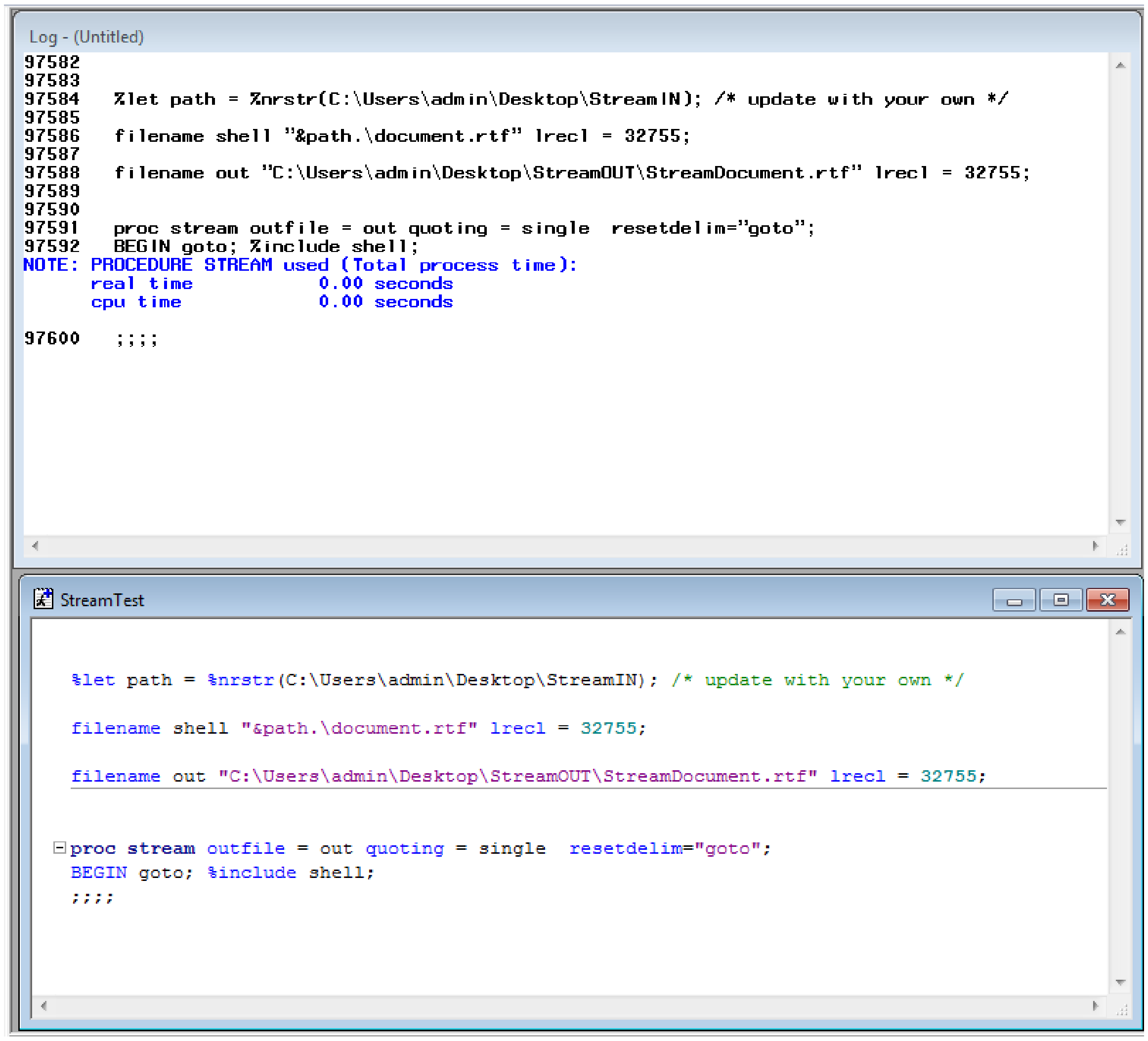
```
97554    data _null_;
97555
97556    %let path = %nrstr(C:\Users\admin\Desktop\StreamIN); /* update with your own */
97557
97558    filename shell "&path.\document.rtf" lrecl = 32755;
97559
97560    %include shell;
97561   +{\rtf1\ansi\ansicpg1252\deff0\deflang1033{\fonttbl{\f0\froman\fprq2\fcharset0 Times New
         -
-
-
-
         180
180
180
180
97561 !+ Roman;}{\f1\fswiss\fprq2\fcharset0 Calibri;}{\f2\fnil\fcharset0 Calibri;}}
97562   +{\*\generator Msftedit 5.41.21.2510;
97562 !+}\viewkind4\uc1\trowd\trgaph108\trleft-108\trrh326\trbrdrl\brdrs\brdrw10
         -
         180
97562 !+\trbrdrt\brdrs\brdrw10 \trbrdrr\brdrs\brdrw10 \trbrdrb\brdrs\brdrw10
97562 !+\trpaddl108\trpaddr108\trpaddfl3\trpaddfr3
ERROR 180-322: Statement is not valid or it is used out of proper order.

97568
97569    run;
```

```
data _null_;

 %let path = %nrstr(C:\Users\admin\Desktop\StreamIN); /* update with your own */

 filename shell "&path.\document.rtf" lrecl = 32755;

 %include shell;

 run;
```

The RTF codes trigger errors and the DATA step get stuck in an endless loop ("DATA STEP running")

(b) The Proc STREAM Solution:

When the same RTF file is run with Proc STREAM, the log becomes error-free:



```
97582
97583
97584   %let path = %nrstr(C:\Users\admin\Desktop\StreamIN); /* update with your own */
97585
97586   filename shell "&path.\document.rtf" lrecl = 32755;
97587
97588   filename out "C:\Users\admin\Desktop\StreamOUT\StreamDocument.rtf" lrecl = 32755;
97589
97590
97591   proc stream outfile = out quoting = single  resetdelim="goto";
97592   BEGIN goto; %include shell;
NOTE: PROCEDURE STREAM used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds

97600   ;;;;
```

```
%let path = %nrstr(C:\Users\admin\Desktop\StreamIN); /* update with your own */

filename shell "&path.\document.rtf" lrecl = 32755;

filename out "C:\Users\admin\Desktop\StreamOUT\StreamDocument.rtf" lrecl = 32755;

proc stream outfile = out quoting = single  resetdelim="goto";
BEGIN goto; %include shell;
;;;;
```

THE PROC STREAM SYNTAX

The Proc STREAM statement specifies an external file with the "OUTFILE=" keyword, as well as options.

The arbitrary text is wrapped inside a "BEGIN" and a four semi-colon ending ";;;;".

There is no "RUN" or "QUIT".

PROC STREAM OUTFILE=*fileref <options>*;

BEGIN

(some text which may contain macro triggers)

;;;;

Two Useful Proc STREAM statement Options:

a. **RESETDELIM=** "label"

The SAS Word Scanner expects macro statements like "%LET" and "%INCLUDE" to begin on a statement boundary -- which means, the statements must be preceded by a semicolon and must end with a semicolon. Therefore to use %LET and %INCLUDE in a Proc STREAM input text, one must place a special marker token before the statements and end the statement with a semicolon. This special marker token is defined with the option RESETDELIM=*label*, where *label* can be any arbitrary SAS name:

```
PROC STREAM  OUTFILE= myfile  RESETDELIM="goto";

BEGIN

goto; %INCLUDE myotherfile;

;;;;
```

The marker token specified by RESETDELIM is also required when a carriage return is required in

```
PROC STREAM  OUTFILE= myfile  RESETDELIM="goto";

BEGIN

Dear Sir, goto NEWLINE;
The profile below is for patient 12345.

;;;;
```

the input text. The keyword "NEWLINE" is used with the marker token:

b. **QUOTING=SINGLE** *(or* **DOUBLE** *or* **BOTH***)*

This option specifies that the single quotation mark (') should be treated like any other character, as expected for: *the patient's blood pressure*.

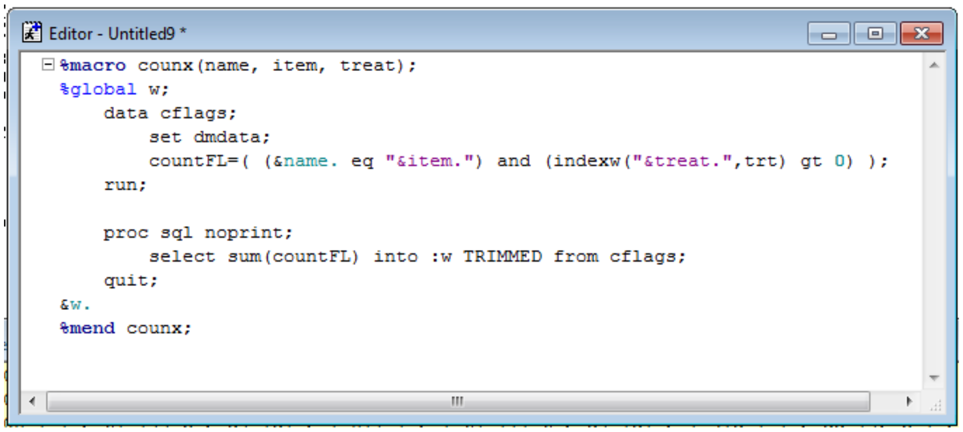
QUOTING=DOUBLE would be required in cases where the text involves, for instance, XML elements in Define-XML documents:

```
<ItemRef ItemOID="ADSL.ARM" OrderNumber="6" Mandatory="No"/>
```

So with Proc STREAM, a macro-embedded RTF document can be placed on the SAS Input Stack without any triggering of errors.

THE PROBLEM WITH MACRO CALLS

Thus, Proc STREAM can allow a macro-embedded RTF document to be processed by the Macro Processor. But there is yet another problem. Putting macro calls in documents becomes problematic if those macros contain DATA steps and/or procedures. Such macros would be resolved by the Macro Processor into the actual constituent DATA step or procedure codes without any execution, and placed back on the Input Stack for streaming out. An example of such macros is shown below:



```
%macro counx(name, item, treat);
%global w;
    data cflags;
        set dmdata;
        countFL=( (&name. eq "&item.") and (indexw("&treat.",trt) gt 0) );
    run;

    proc sql noprint;
        select sum(countFL) into :w TRIMMED from cflags;
    quit;
&w.
%mend counx;
```

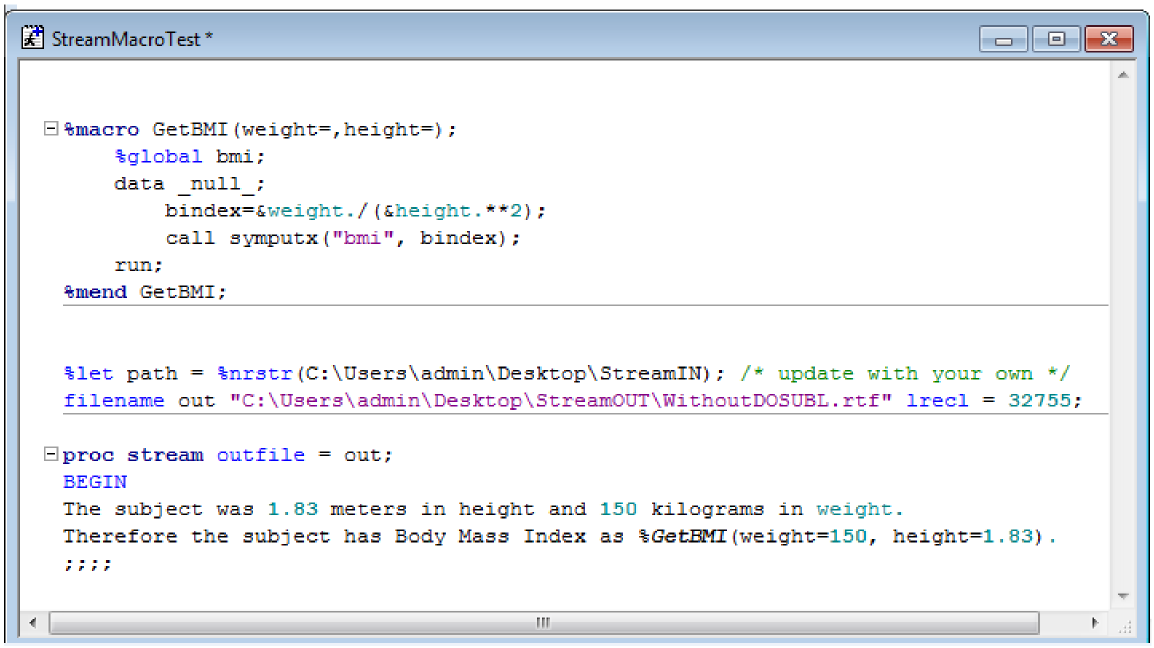
We desire the analysis macro to generate a single value.

However when %counx() is called, the macro processor would return the generated code instead of the computed value.

THE NEED FOR DOSUBL AND %SYSFUNC

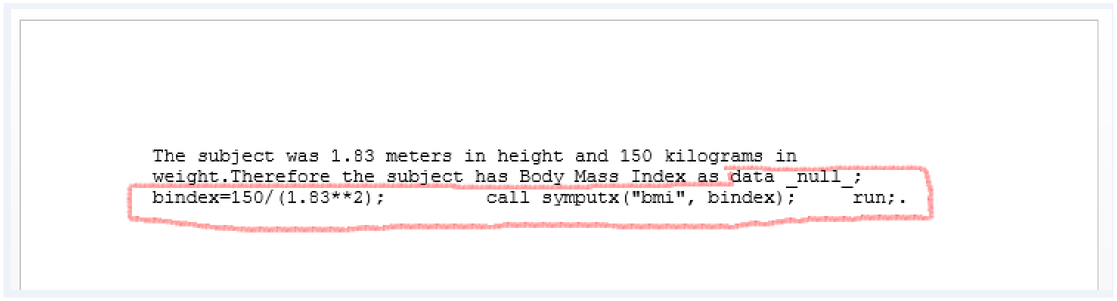
The DOSUBL function enables the immediate execution of SAS code after a text string is passed. If the text is a DATA step or a PROC program, DOSUBL would allow them to execute and wait for them to complete. Also any macro variables that are created or updated during the execution of the submitted code are exported back to the calling environment. DOSUBL does not return until the execution of the SAS code is fully completed.
Thus if DOSUBL is called by a macro via %SYSFUNC, then the macro is forced to execute the DATA step and Proc SQL codes and any macro variable created inside the DOSUBL becomes available, instead of the macro execution returning the raw lines of DATA step or Proc codes to the input stack. %SYSFUNC forces execution of functions it carries as arguments. Also for proper timing of resolution for macro variables enclosed in quotes, the outermost quotes inside the DOSUBL function should be single. Below is an example with a macro that calculates the Body Mass Index of a subject, as processed by Proc STREAM:

(a) Without DOSUBL:



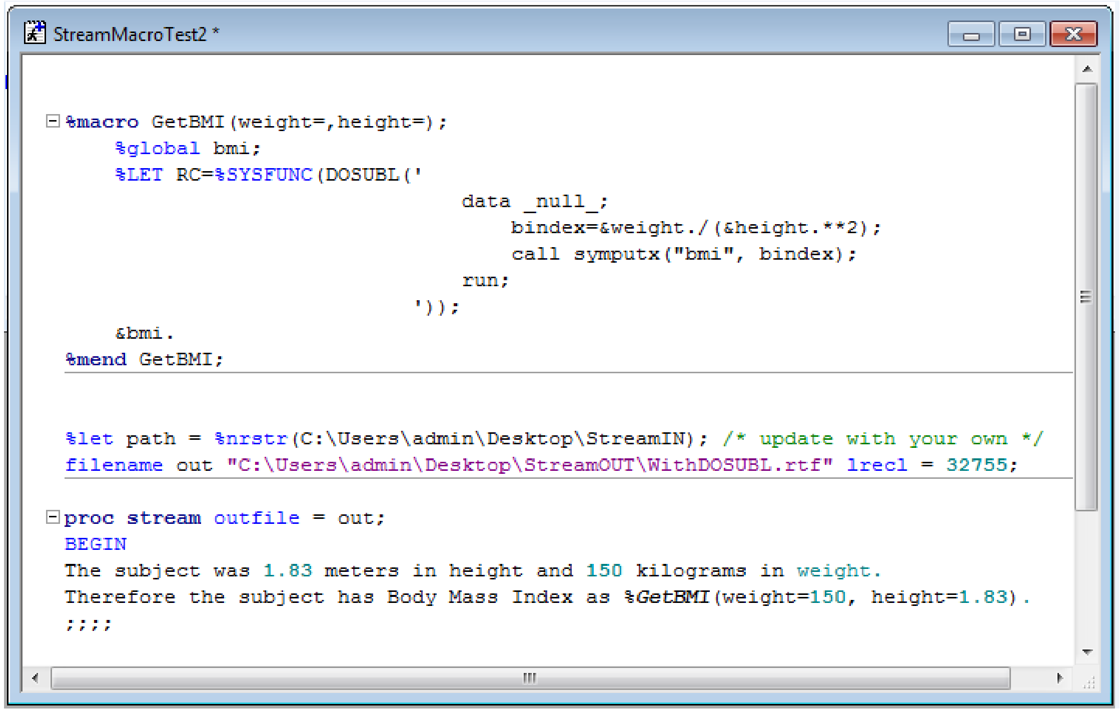
```

%macro GetBMI(weight=,height=);
    %global bmi;
    data _null_;
        bindex=&weight./(&height.**2);
        call symputx("bmi", bindex);
    run;
%mend GetBMI;


%let path = %nrstr(C:\Users\admin\Desktop\StreamIN); /* update with your own */
filename out "C:\Users\admin\Desktop\StreamOUT\WithoutDOSUBL.rtf" lrecl = 32755;

proc stream outfile = out;
BEGIN
The subject was 1.83 meters in height and 150 kilograms in weight.
Therefore the subject has Body Mass Index as %GetBMI(weight=150, height=1.83).
;;;;
```

The streamed output shows the DATA step code instead of the calculated BMI value:



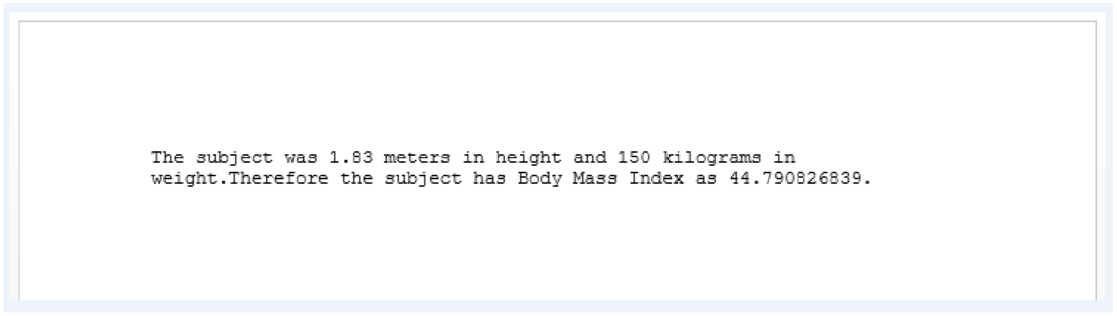
```
The subject was 1.83 meters in height and 150 kilograms in
weight.Therefore the subject has Body Mass Index as data _null_;
bindex=150/(1.83**2);         call symputx("bmi", bindex);     run;.
```

(b) With DOSUBL:



```
%macro GetBMI(weight=,height=);
    %global bmi;
    %LET RC=%SYSFUNC(DOSUBL('
                            data _null_;
                                bindex=&weight./(&height.**2);
                                call symputx("bmi", bindex);
                            run;
                        '));
    &bmi.
%mend GetBMI;


%let path = %nrstr(C:\Users\admin\Desktop\StreamIN); /* update with your own */
filename out "C:\Users\admin\Desktop\StreamOUT\WithDOSUBL.rtf" lrecl = 32755;

proc stream outfile = out;
BEGIN
The subject was 1.83 meters in height and 150 kilograms in weight.
Therefore the subject has Body Mass Index as %GetBMI(weight=150, height=1.83).
;;;;
```

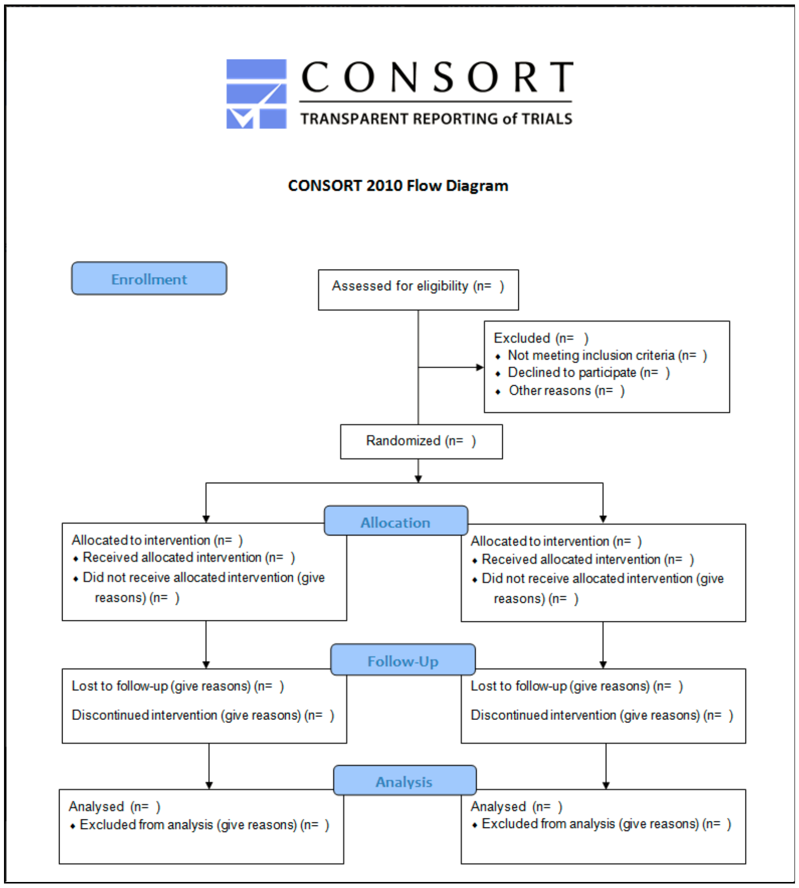
Now, the DATA step code is fully resolved into a value:



```
The subject was 1.83 meters in height and 150 kilograms in
weight.Therefore the subject has Body Mass Index as 44.790826839.
```

APPLICATION TO THE CONSORT FLOW DIAGRAM

The acronym CONSORT stands for "CONsolidated Standards Of Reporting Trials" and includes a flow diagram which shows the flow of participants through each stage of a clinical trial as shown below:


CONSORT
TRANSPARENT REPORTING of TRIALS
CONSORT 2010 Flow Diagram
Enrollment
Assessed for eligibility (n=)
Excluded (n=)
Not meeting inclusion criteria (n=)
Declined to participate (n=)
Other reasons (n=)
Randomized (n=)
Allocation
Allocated to intervention (n=)
Received allocated intervention (n=)
Did not receive allocated intervention (give reasons) (n=)
Allocated to intervention (n=)
Received allocated intervention (n=)
Did not receive allocated intervention (give reasons) (n=)
Follow-Up
Lost to follow-up (give reasons) (n=)
Discontinued intervention (give reasons) (n=)
Lost to follow-up (give reasons) (n=)
Discontinued intervention (give reasons) (n=)
Analysis
Analysed (n=)
Excluded from analysis (give reasons) (n=)
Analysed (n=)
Excluded from analysis (give reasons) (n=)

Specifically, the flow diagram includes the number of participants assessed for potential enrollment into the trial (if known) and the number excluded at this stage either because they did not meet the inclusion criteria or declined to participate. The diagram also provides for each intervention group the numbers of participants who were randomly assigned, received treatment as allocated, completed treatment as allocated, and were included in the main analysis, with numbers and reasons for exclusions at each step.

Usually, the counts in the diagram are manually typed in. But since 2011, various attempts[2-6] have been made to automate the process. In two methods[3,5], SAS was used to read the RTF text strings including codes, and the appropriate RTF codes replaced with codes containing the count values, using the SAS function TRANSTRN. With that approach, a macro could compute and insert the counts directly into the RTF codes. In another series of methods[2], a Visual Basic script (VBScript) is used to do a find-and-replace operation, where place-holders in the document are substituted with SAS-generated counts programmatically. More elaborate approaches have also been described[2], using VB scripting in addition to the Annotate Facility of SAS Graph and the Windows Scripting Host. However all such techniques as described above can be quite challenging for a programmer not well-versed in scripting languages.

With the technique proposed in this paper, the entire process of automatically populating a CONSORT flow diagram is extremely simplified: simple macros are created for producing counts, the macro calls are typed into the CONSORT template, and the template in RTF format submitted to the SAS Input Stack for macro processing via Proc STREAM, which then streams back the CONSORT template as-is except now, populated automatically with counts.

STRATEGY

1. Create counting macros for the various sections of the flow diagram.
2. Obtain a copy of the CONSORT Flow diagram template in Microsoft WORD format.
3. Create a smart version by typing counting macro calls after the “N=” parts of the text boxes.
4. Convert the smart CONSORT Flow diagram from .DOC to .RTF format (call it “SmartConsort.rtf”).
5. Write Proc STREAM code with %include SmartConsort.rtf and output file as “FilledConsortFlowDiagram.rtf”
6. Run main program to define analysis macros and to execute Proc STREAM.
7. Open FilledConsortFlowDiagram.rtf with Microsoft WORD.

AN EXAMPLE USING BINARY PROGRAMMING FOR COUNTING

1. CREATE BINARY FLAGS AND COMPUTE THE SUMS WITH PROC SQL:

(a) Read Input Dataset:

--------------------------------EXPLANATION OF FLAGS------------------------

ENROLLMENT:	Screened for Eligibility:	SCREENFL=[not missing(ENRFL)]
	Excluded:	EXCLUDFL=[ENRLFL=0]
	Not meet inclusion criteria:	[EXCRITFL=1]
	Declined to participate:	[EXDECLFL=1]
	Other reason:	[EXOTHRFL=1]
	Randomized:	[RANDFL=1]
ALLOCATION:	Allocated to Experimental Group:	[ACTFL=1]
	Received allocated intervention:	[RECACTFL=1]
	Did not receive allocated intervention:	NORACTFL=[RECACTFL=0]
	Allocated to Control Group:	[PBOFL=1]
	Received allocated intervention:	[RECPBOFL=1]
	Did not receive allocated intervention:	NORPBOFL=[RECPBOFL=0]
FOLLOW-UP:	Lost to follow-up (experimental group):	[FUPACTFL=1]
	Discontinued treatment (experimental group):	DSCACTFL=1]
	Lost to follow-up (control group):	[FUPPBOFL=1]
	Discontinued treatment (control group):	[DSCPBOFL=1]
ANALYSIS:	Analyzed (experimental group):	[ANLACTFL=1]
	Excluded from analysis (experimental group):	NALACTFL=[ANLACTFL=0]
	Analyzed (control group):	[ANLPBOFL=1]
	Excluded from analysis (control group):	NALPBOFL=[ANLPBOFL=0]

```
*===============================================================
C R E A T E   I N P U T   D A T A S E T
==============================================================;
data adsl;
infile datalines;
input USUBJID $ ENRLFL EXCRITFL EXDECLFL EXOTHRFL RANDFL ACTFL RECACTFL PBOFL
RECPBOFL FUPACTFL DSCACTFL FUPPBOFL DSCPBOFL ANLACTFL ANLPBOFL;
SCREENFL=not missing(ENRLFL);
EXCLUDFL=(ENRLFL eq 0);
NORACTFL=(ACTFL eq 1) AND (RECACTFL eq 0);
NORPBOFL=(PBOFL eq 1) AND (RECPBOFL eq 0);
NALACTFL=(ACTFL eq 1) AND (ANLACTFL eq 0);
NALPBOFL=(PBOFL eq 1) AND (ANLPBOFL eq 0);

datalines;
ABC1230001 1 0 0 0 1 1 1 0 0 0 0 0 0 1 0
ABC1230002 1 0 0 0 1 1 1 0 0 0 0 0 0 1 0
ABC1230003 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
ABC1230004 1 0 0 0 1 0 0 1 1 0 0 0 0 0 1
ABC1230005 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
ABC1230006 1 0 0 0 1 1 1 0 0 0 0 0 0 1 0
ABC1230007 1 0 0 0 1 0 0 1 1 0 0 0 0 0 1
ABC1230008 1 0 0 0 1 1 1 0 0 0 0 0 0 1 0
ABC1230009 1 0 0 0 1 0 0 1 1 0 0 0 0 0 1
ABC1230010 1 0 0 0 1 1 1 0 0 0 0 0 0 1 0
ABC1230011 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
ABC1230012 1 0 0 0 1 0 0 1 1 0 0 0 0 0 1
ABC1230013 1 0 0 0 1 1 1 0 0 0 0 0 0 1 0
ABC1230014 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
ABC1230015 1 0 0 0 1 0 0 1 1 0 0 0 0 0 1
ABC1230016 1 0 0 0 1 1 1 0 0 0 0 0 0 1 0
ABC1230017 1 0 0 0 1 0 0 1 1 0 0 1 0 0 0
ABC1230018 1 0 0 0 1 1 1 0 0 0 0 0 0 1 0
ABC1230019 1 0 0 0 1 0 0 1 1 0 0 0 0 0 1
ABC1230020 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
ABC1230021 1 0 0 0 1 1 1 0 0 0 0 0 0 1 0
ABC1230022 1 0 0 0 1 0 0 1 1 0 0 0 1 0 0
;
run;

%macro cx(flag);
%global rx;
%let u1=%sysfunc(dosubl('proc sql noprint; select sum(&flag.) into :rx
TRIMMED from adsl;quit;'));
&rx.
%mend cx;

%let path = %nrstr(C:\Users\admin\Desktop\StreamIN); /* update with your own
*/

filename shell "&path.\SmartConsortFlowDiagramRTF.rtf" lrecl = 32755;

filename out "C:\Users\admin\Desktop\StreamOUT\ConsortFlowDiagram.rtf" lrecl
= 32755;

proc stream outfile = out quoting = single  resetdelim="goto";
BEGIN goto; %include shell;
```

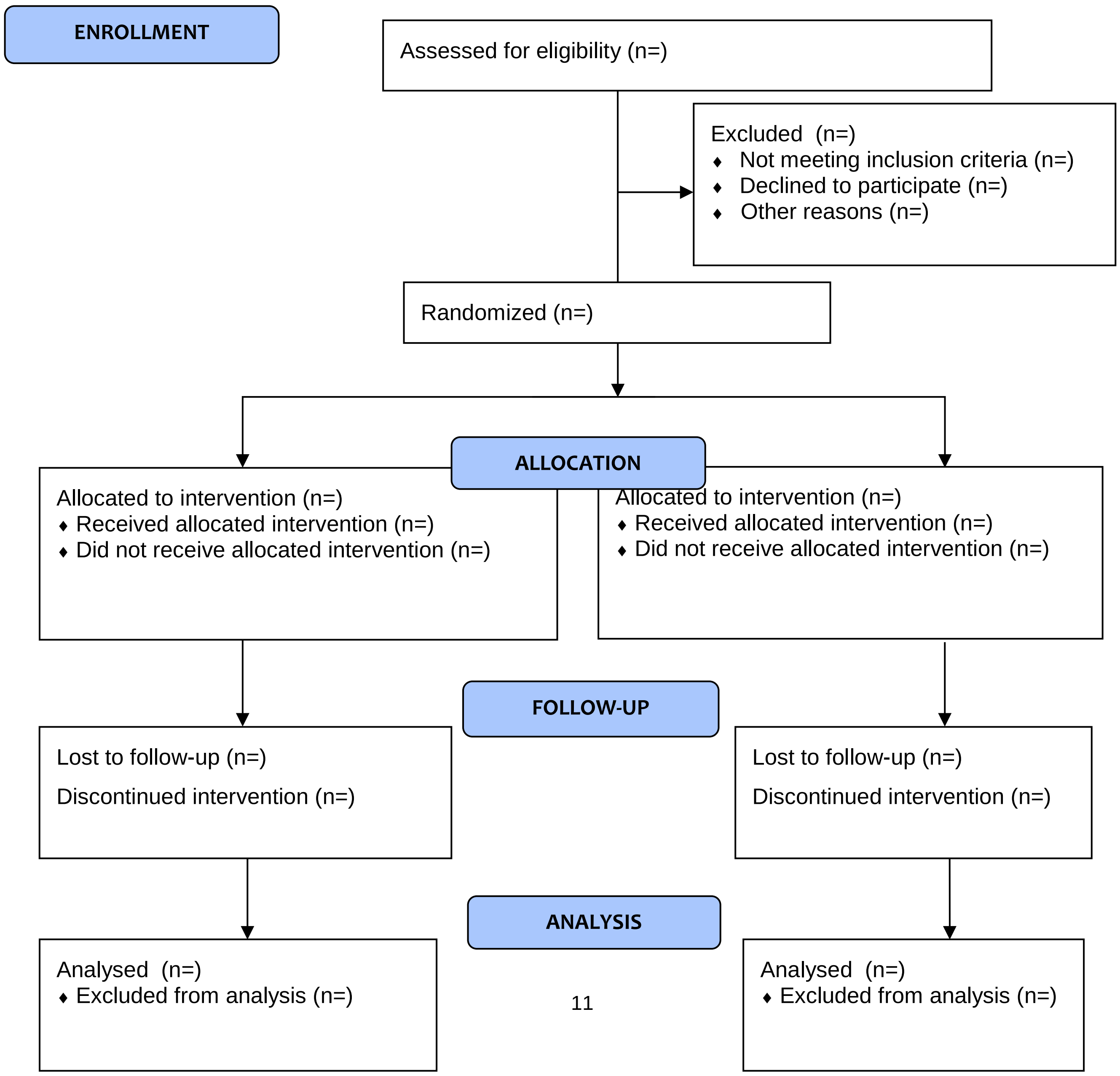
; ; ; ;

2. OBTAIN A CONSORT FLOW DIAGRAM TEMPLATE:

A CONSORT Flow Diagram template in MS WORD format can be downloaded from various websites[1], as shown below:

CONSORT
TRANSPARENT REPORTING of TRIALS

CONSORT 2010 Flow Diagram

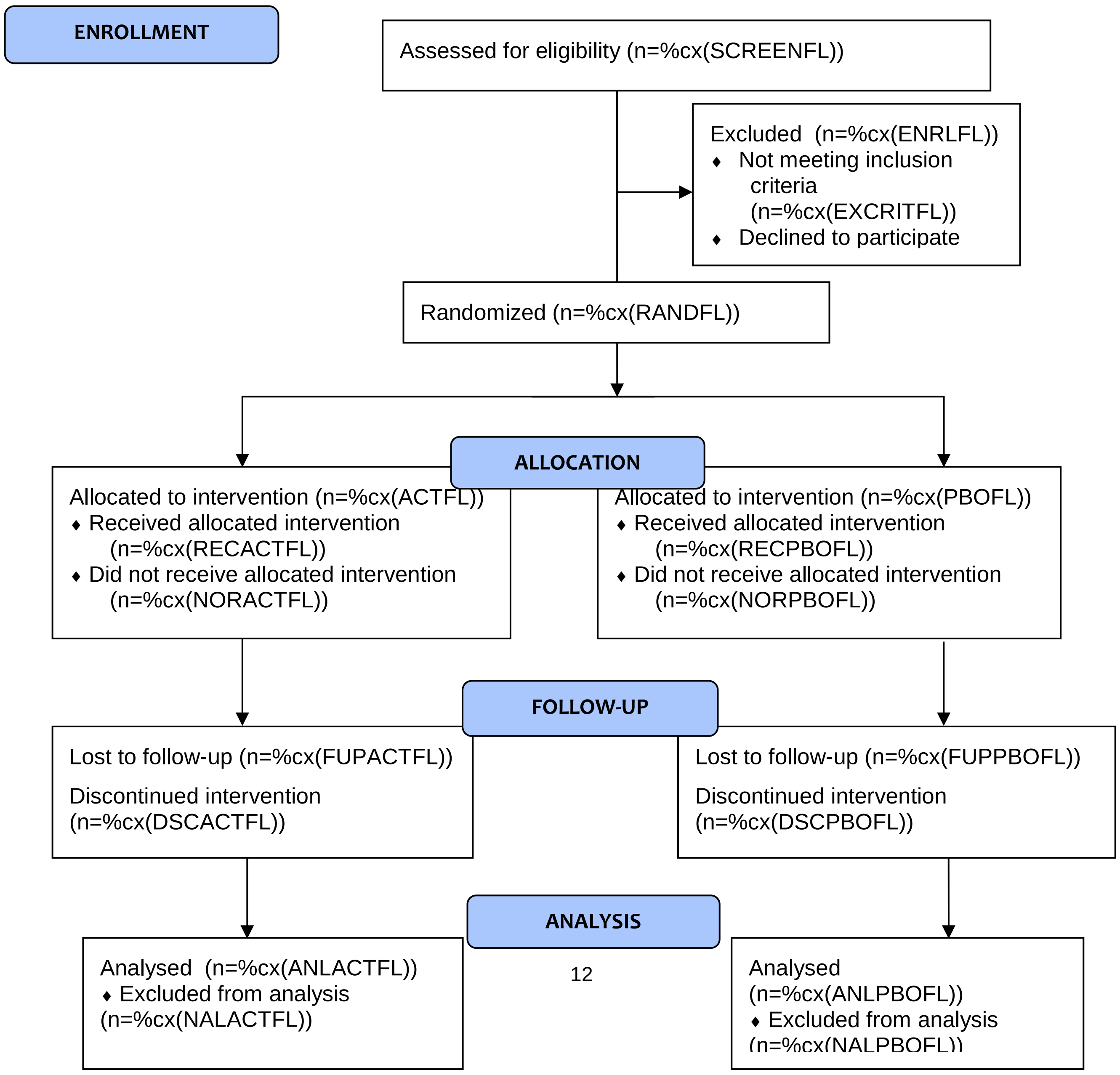

ENROLLMENT
Assessed for eligibility (n=)
Excluded (n=)
♦ Not meeting inclusion criteria (n=)
♦ Declined to participate (n=)
♦ Other reasons (n=)
Randomized (n=)
ALLOCATION
Allocated to intervention (n=)
♦ Received allocated intervention (n=)
♦ Did not receive allocated intervention (n=)
Allocated to intervention (n=)
♦ Received allocated intervention (n=)
♦ Did not receive allocated intervention (n=)
FOLLOW-UP
Lost to follow-up (n=)
Discontinued intervention (n=)
Lost to follow-up (n=)
Discontinued intervention (n=)
ANALYSIS
Analysed (n=)
♦ Excluded from analysis (n=)
Analysed (n=)
♦ Excluded from analysis (n=)

The above template then gets typed in with counting macro calls at the "n=" locations in the text boxes to become the "SmartConsortFlowDiagram".doc, as shown below "Smart:

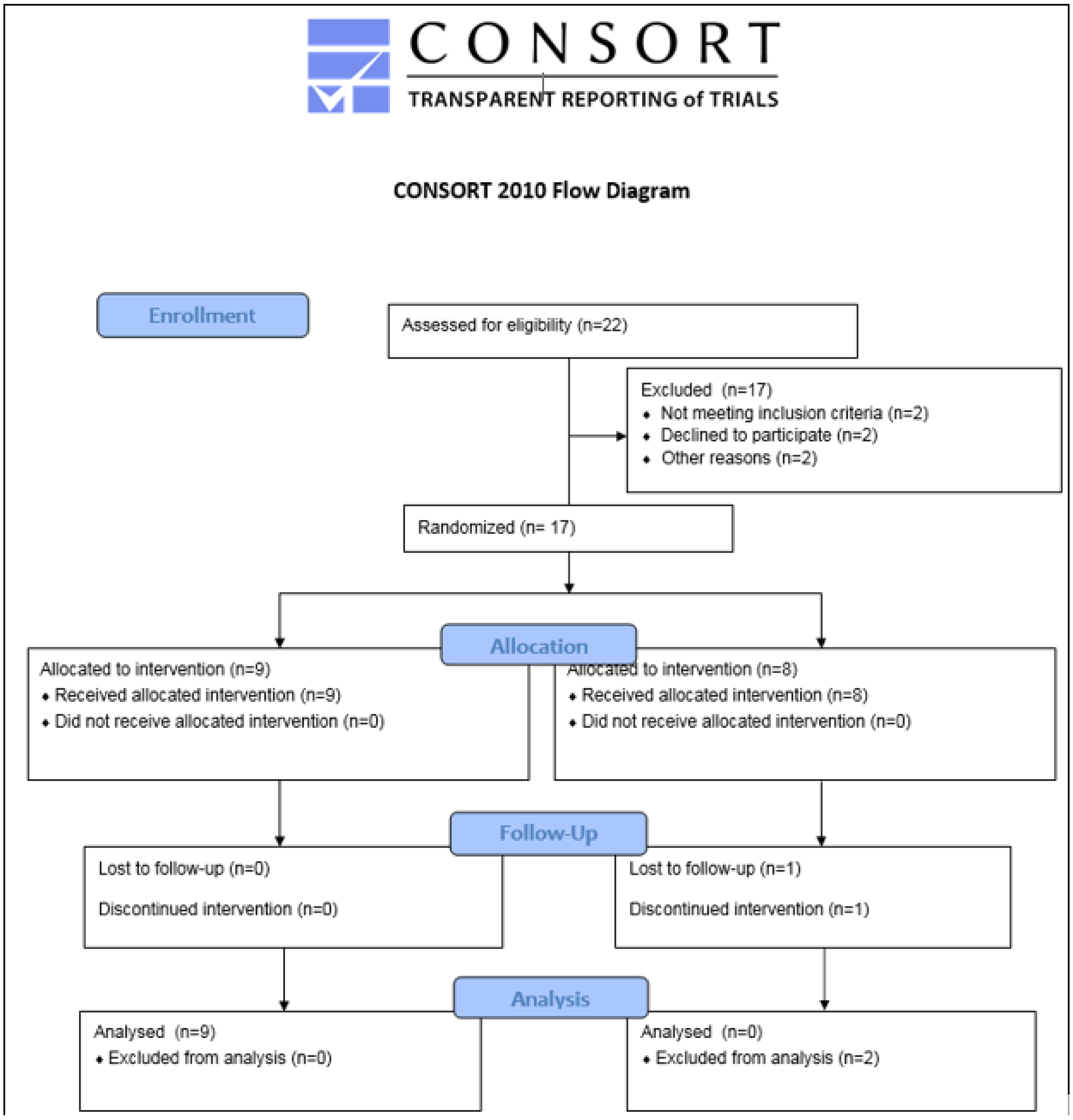
CONSORT

TRANSPARENT REPORTING of TRIALS

CONSORT 2010 Flow Diagram


ENROLLMENT
Assessed for eligibility (n=%cx(SCREENFL))
Excluded (n=%cx(ENRLFL))
♦ Not meeting inclusion criteria (n=%cx(EXCRITFL))
♦ Declined to participate
Randomized (n=%cx(RANDFL))
ALLOCATION
Allocated to intervention (n=%cx(ACTFL))
♦ Received allocated intervention (n=%cx(RECACTFL))
♦ Did not receive allocated intervention (n=%cx(NORACTFL))
Allocated to intervention (n=%cx(PBOFL))
♦ Received allocated intervention (n=%cx(RECPBOFL))
♦ Did not receive allocated intervention (n=%cx(NORPBOFL))
FOLLOW-UP
Lost to follow-up (n=%cx(FUPACTFL))
Discontinued intervention (n=%cx(DSCACTFL))
Lost to follow-up (n=%cx(FUPPBOFL))
Discontinued intervention (n=%cx(DSCPBOFL))
ANALYSIS
Analysed (n=%cx(ANLACTFL))
♦ Excluded from analysis (n=%cx(NALACTFL))
Analysed (n=%cx(ANLPBOFL))
♦ Excluded from analysis (n=%cx(NALPBOFL))

And after being processed with Proc STREAM, an output in RTF is streamed out to a designated output folder, and the file opened with MS WORD, as shown below:


CONSORT
TRANSPARENT REPORTING of TRIALS
CONSORT 2010 Flow Diagram
Enrollment
Assessed for eligibility (n=22)
Excluded (n=17)
Not meeting inclusion criteria (n=2)
Declined to participate (n=2)
Other reasons (n=2)
Randomized (n= 17)
Allocation
Allocated to intervention (n=9)
Received allocated intervention (n=9)
Did not receive allocated intervention (n=0)
Allocated to intervention (n=8)
Received allocated intervention (n=8)
Did not receive allocated intervention (n=0)
Follow-Up
Lost to follow-up (n=0)
Discontinued intervention (n=0)
Lost to follow-up (n=1)
Discontinued intervention (n=1)
Analysis
Analysed (n=9)
Excluded from analysis (n=0)
Analysed (n=0)
Excluded from analysis (n=2)

CAVEATS

(1) Because RTF (or WordPad) can't display text boxes, for Flow Diagrams Macro Calls must first be typed on the MS WORD version of the CONSORT template before saving as RTF.

(2) Macro Calls MUST be typed, and not copied from an external source, to ensure the macro syntax is not contaminated with extraneous RTF codes.

(3) It's okay to copy Macro Calls from one part of the WORD document to another part of the same WORD document.

(4) The Macro Calls MUST be plain font (no color or bold, or italic), to avoid syntax contamination with extraneous RTF codes.

(5) The outermost quotes of the DOSUBL argument MUST be single.

(6) Creating macro variables with Proc SQL requires the TRIMMED keyword to avoid leading spaces before macro variable values.

(7) The macro variable statement inside the macros should NOT end with a semicolon.

Example:
```
      %macro bn(gp);
      %let t=%sysfunc(dosubl('proc sql noprint;select
count(distinct subjid) into :bgn TRIMMED from dmdata where
indexw("&gp.",trt) gt 0;quit;'));
      &bgn.
      %mend bn;
```

REFERENCES

(1) http://www.consort-statement.org/consort-statement/flow-diagram

(2) Fairfield-Carter, Brian and Suzanne Humphreys, 2011, "Alternative Approaches to Creating Disposition Flow Diagrams", Proceedings of the Pharmaceutical SAS Users Group Conference (PharmaSUG), Paper TS10.
http://www.lexjansen.com/phuse/2011/ts/TS10.pdf

(3) Carpenter, Art and Fisher, Dennis, 2012, "Reading and Writing RTF documents as Data: Automatic Completion of CONSORT Flow Diagrams." Proceedings of the Pharmaceutical SAS Users Group Conference (PharmaSUG), Paper TF16
http://www.pharmasug.org/proceedings/2012/TF/PharmaSUG-2012-TF16.pdf

(4) Abbott, David H., 2013, "Computing Counts for CONSORT Diagrams: Three Alternatives", The SouthEast SAS Users Group (SESUG), Paper CC-11-2013
http://analytics.ncsu.edu/sesug/2013/BtB-12.pdf

(5) Mallavarapu, Anusha and Shults, Dean, 2016, "CONSORT Diagram: Doing it with SAS". Pharmaceutical Users Software Exchange (PhUSE) Poster PP03 http://www.lexjansen.com/phuse/2016/pp/PP03.pdf

CONTACT INFORMATION

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

Joseph W. Hinson, PhD
inVentiv Health
202 Carnegie Center, Suite 200
Princeton, NJ, 08540
1-609-282-1615
joehinson@outlook.com