PharmaSUG 2012 - Paper AD15

Database Tectonics: Assessing Drift in Analysis Data

Brian Fairfield-Carter, ICON Clinical Research, Redwood City, CA

Jasmin Fredette, ICON Clinical Research, Redwood City, CA

ABSTRACT

In long-term projects, particularly those with separate analyses conducted at major study milestones (i.e. at the end of acute-treatment and long-term extension phases), project documentation needs to account for changes to the analysis data resulting from evolving data standards, as well as those resulting from changes to the underlying raw data. The database at the time of the final/extension lock includes records from acute-phase visits, and because of 'data drift' (changes to raw data and to derivation logic) these acute-phase records may differ from those at the time of the acute lock. Since they may ultimately effect statistical inference, these differences need to be documented (often retrospectively), but the process is complicated because (1) differences resulting from changes to the raw data are confounded with those resulting from changes to data-handling rules and derivation logic, and (2) differences in analysis data accumulate across dependency layers (i.e. where lower-level analysis datasets are used in the creation of higher-level analysis datasets).

This paper describes 'cross-comparison', where acute-lock programs are run on extension-lock raw data and extension-lock programs on acute-lock raw data, and the resulting datasets compared back against the original acute-lock analysis datasets, as a mechanism for distinguishing between differences arising from changes to program logic from those arising from changes to raw data. The use of a simple 'pre-processor' in implementing cross-comparison is illustrated, since the technique is 'retro-fitted' to an existing project and by 'augmenting' SAS code at run-time, the task of copying and modifying legacy code can be avoided.

INTRODUCTION

Most people will recall from basic Geography the concepts of 'continental drift' and 'plate tectonics', describing the mechanism whereby what was once a continuous landmass divided into the continents we know today.

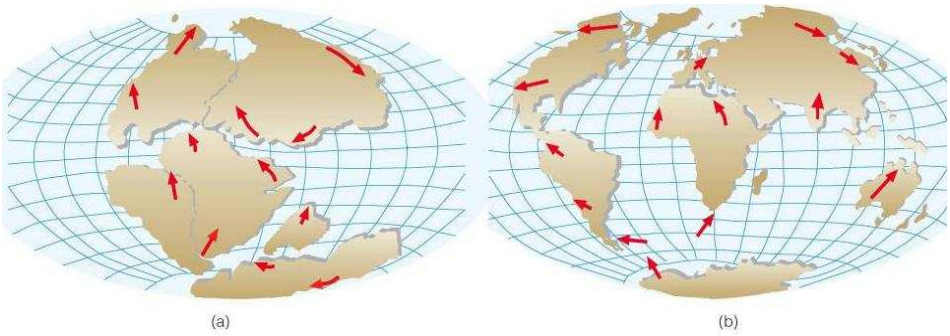

Figure 1. Plate Tectonics and Continental Drift

If we consider a database, and in particular one comprised of a collection of analysis datasets, there's a decent analogy to be made: the datasets can be of a size and complexity that seems 'continental' in scale and, paradoxically when we consider that most adhere to some form of 'data standards', can change over time in their content, structure, relationships, and underlying rules. In long-term projects, data can 'drift', and owing to the hierarchical nature of analysis datasets, this drift can produce artifacts whose origins are obscure and difficult to document.

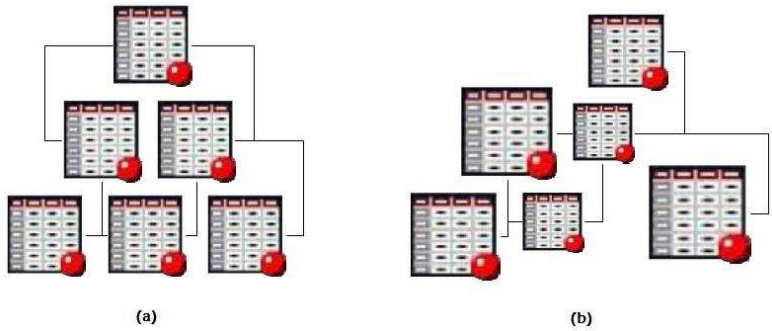

Figure 2. Database Tectonics and Data Drift

Figure 3 shows the timeline for a 'case study': a typical long-term project, consisting of an acute-treatment phase and a long-term extension phase:

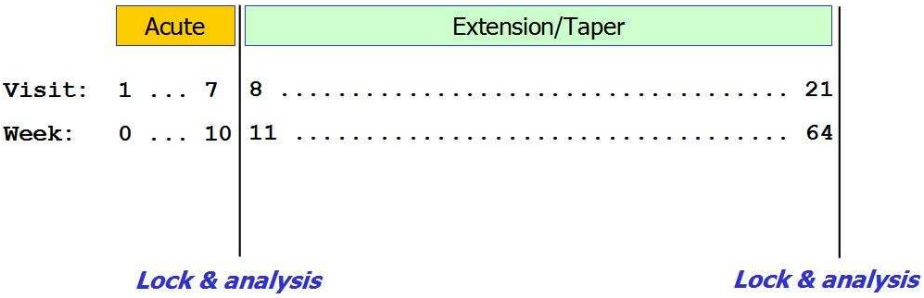

Figure 3. Timeline for a typical long-term study

This 'case study' will be used to provide context for this discussion, and consists more precisely of a double-blind acute-treatment phase followed by an open-label extension phase. At the end of the acute-treatment phase the database (including the analysis datasets) is locked and the acute-phase analysis is run. Patients are then enrolled into the long-term extension phase, and at the end of this phase the database is again locked, and the extension-phase analysis is run. Much of the extension-phase statistical output pools records from both study phases, and submission to the FDA ultimately includes both versions of the acute-phase data, one from each of the major study milestones. The question then arises:

> ***Would the same conclusions be drawn from an analysis of the acute-phase locked data as from an analysis of the set of acute-phase records that reside in the locked database at the end of the extension phase?***

In other words, did acute-phase analysis data change during the course of the extension phase, and if so, in what way and how significantly? In an ideal world, this would not be an issue, but in actuality, and in the 'case study' presented here, changes do take place in the acute-phase analysis data during the course of the extension phase. These changes arise from two sources: the raw data (new queries are issued on acute-phase records, and additional acute-phase records are introduced into the raw data), and derivation programs (computational methods and data-handling rules are revised). Without question, both of these changes are justified (all identifiable corrections do need to be made to raw data, and there are often good reasons for modifying data-handling rules, like when new and unexpected conditions arise in the raw data), but their cumulative effect on the analysis data needs to be assessed and documented, ultimately to address if and why fundamental conclusions might differ between major study milestones.

LOGISTICAL PROBLEMS IN DOCUMENTING DATA DRIFT

What is it that can make documenting these changes a non-trivial task? Can we simply run the COMPARE procedure on the extension-lock datasets against the archived acute-lock datasets? Is it sufficient to only review changes in raw data? Ultimately it depends on the nature and extent of changes that have taken place.

1. ANALYSIS DATASETS ARE HIERARCHICAL

For a start, analysis datasets may consist of several 'dependency layers', where lower-level datasets are used in the creation of higher-level datasets. For example, a lower-level dataset recording study drug dosing may be used to

create a higher-level dataset summarizing treatment compliance and overall exposure. While changes that took place in the lower-level dataset may be obvious and easy to describe, they may manifest in more obscure differences in the higher-level datasets. This is why, even when no changes to derivation code take place between database lock events, it may be insufficient to simply document changes that took place in raw data.

2. ROOT CAUSES MAY BE CONFOUNDED

Let's say we compare acute-lock analysis datasets to the acute-phase records present in the extension-lock analysis datasets. Recall that in our 'case study', changes took place both in the raw data and in derivation code, so when these changes are translated into differences in analysis data, can we distinguish between the root causes? In statistics, a 'confounding' variable is one that is correlated with both the dependent variable and another independent variable; the concept applies here as well, since the effects of changes to raw data and to derivation code are 'confounded' when we look at their effects on analysis data. If we simply look at differences between two versions of a given analysis dataset, we can't necessarily tell which differences result from changes to raw data and which result from changes to program logic.

3. RECORD-COUNT DIFFERENCES TEND TO HIDE VALUE-LEVEL DIFFERENCES

When a specific data value changes (for example, a specific lab result value is queried and changed), we would refer to this as producing a 'value-level difference'. Taken in isolation, these differences are relatively easy to explain. However, some analysis datasets include 'generated' or 'imputed' records, for instance between bounding dates. A good example is a study drug dosing dataset where dosing records are 'imputed' based on the dates of dispensing visits. If rules and assumptions surrounding the handling of missing dispensing dates change, this will result in changes to the number of dosing records generated, and hence will produce record-count differences. These record-count differences can be problematic since they tend to hide underlying value-level differences.

PURPOSE

Using the case study as an example, and in light of these logistical considerations, this paper seeks to address the following questions:

- How do we document differences between versions of an analysis database?
- How do we distinguish between differences arising from changes to raw data and those arising from changes to programming logic?
- How do we efficiently implement these comparisons, particularly when the requirement was not anticipated early on in study conduct?

'CROSS-COMPARISON' AS A GENERALIZED SOLUTION

In statistics, we apply the concept of 'control' in dealing with confounded variables. In other words, we hold one factor constant while another is permitted to vary, in order to look at the effect on the dependent variable of changes to a single independent variable. In dealing with confounded causes of changes to analysis data in our case study, we can apply the same principle, termed here 'cross-comparison'. In essence, we first remove changes to raw data as a source of changes to analysis data, so that we can see the results of changes to program logic in isolation, and then we do the reverse – remove changes to program logic as a source of changes to analysis data and look at the results of changes to raw data in isolation. This is shown schematically in Figure 4:

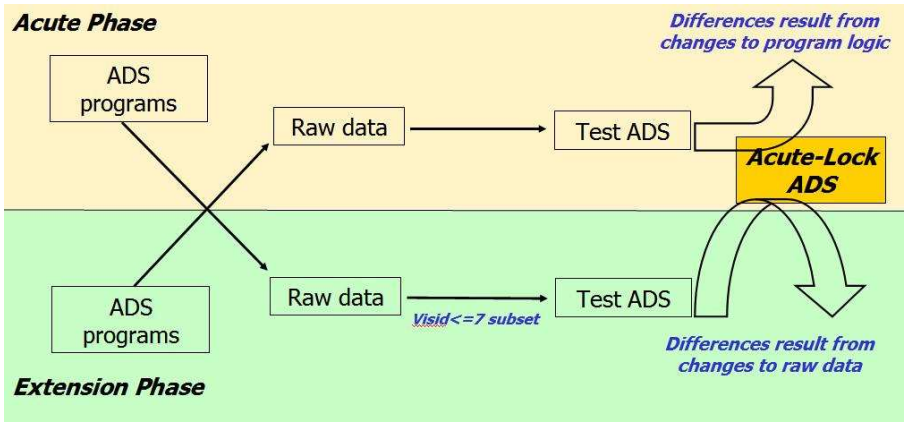

Figure 4. Cross-comparison (ADS=Analysis Datasets)

In one 'branch' of the cross-comparison, derivation programs as they stood at the time of the extension-phase lock are run not on extension-lock raw data, but rather on acute-lock raw data. This essentially removes raw data as a potential cause of variation in analysis datasets, meaning that when the resulting 'test' analysis datasets are compared against the original acute-lock analysis datasets, differences that are observed must be the result of changes to program logic.

In the other branch of the cross-comparison, derivation programs as they stood at the time of the acute-phase lock are run on extension-lock raw data (subset to only include acute-phase visits). This effectively removes changes to program logic as a potential cause of variation in analysis datasets, meaning that when the resulting 'test' datasets are compared against the original acute-lock datasets, differences that are observed must be the result of changes to raw data.

FINDINGS AND DOCUMENTATION

More 'conventional' programming tasks, such as programming a table or an analysis datasets, tend to have a clearly-defined endpoint. The 'final product' of this cross-comparison is less obvious. In our case study, the most basic documentation consists of the raw PROC COMPARE output resulting from the comparisons between 'test' analysis datasets and the archived acute-lock analysis datasets. However, in some cases this raw output is not particularly meaningful, especially when there are record-count differences. Ultimately, we may have to take datasets one at a time, and in some cases write additional ad hoc code in order to dig further into specific discrepancies: this may involve tracing the origins of record-count differences, and may be enhanced through the use of 'diff' utilities in comparing versions of derivation programs. In the end the task is fairly open-ended, and involves a certain amount of subjectivity in deciding which discrepancies require more extensive investigation, though the overarching question is always whether or not the discrepancy could potentially contribute to a contradiction between the conclusions drawn at major study milestones.

To illustrate, here are a couple of database-comparison findings from our case study:

1. Introduction of 'new' Acute-phase ECG records

Patient '123' showed new unscheduled Acute-phase ECG records in the extension-lock database, assigned unscheduled visit ID 1.01. When unscheduled lab records were then 'sequenced' so that unscheduled visit ID matched the overall chronological order of unscheduled assessments, it meant that lab records that had previously been assigned visit ID 1.01 were now assigned visit ID 1.02 (since the date of this unscheduled lab assessment fell after the visit 1.01 ECG assessment). The change to the ECG analysis dataset was easily shown to result from the new ECG records in the raw data, but the causes of the secondary effects on the LAB analysis dataset were less obvious (they were shown to result from changes to raw data, but not to changes to raw *lab* data). Ultimately though, changes to visit ID in the lab data had no effect on the selection of baseline values, and hence resulted in no changes to the analysis.

2. Changes to data-handling rules for unscheduled dispensing

Dosing in this study was assumed to take place daily between scheduled dispensing visits, but when unscheduled dispensing took place, with a changed dose level, it wasn't clear whether the new dose level was necessarily administered right away, or if it took effect after the tablets dispensed at the earlier visit had been exhausted. This 'apparent dose level' has implications both to estimates of total/cumulative drug exposure, as well as to various correlative safety measures, such as dose level at the time of Adverse Event onset. Changes to the handling of

unscheduled dispensing naturally resulted in changes to the 'study treatment' analysis dataset, but also resulted in changes to higher-order datasets capturing dosing compliance and overall exposure, as well as to any datasets capturing dose level at the time of assessment.

IMPLEMENTING CROSS-COMPARISON USING A PRE-PROCESSOR

Running this sort of cross-comparison might seem like a trivial matter, but it really depends on the design of the programming environment, and whether or not the requirement was anticipated at the time the programming environment was set up. In the case study, and as is probably the case with many stats programming environments, the raw-data and analysis-data libraries, and format catalogs and autocall libraries were assumed to be in sub-directories under a common project directory (in fact, in the case study these references are set via auto-detection of the root directory), with separate archive folders holding snap-shots at key project milestones. The result was that it was not entirely simple to run archived acute-lock analysis programs against extension-lock raw data, but rather demanded the modification of this archived code. The question was, under the circumstances what were the options to, for instance, copying 8+ Gb of data, and/or copying and modifying in the order of 80 derivation programs in order to deal with these inherent root-path dependencies?

As it happened, the problem offered a practical application of a SAS-based 'pre-processor' (Fairfield-Carter, 2010); the objective of not copying or modifying any program source files, or moving any data around, essentially requires that programs be modified at run-time, which is exactly the niche filled by a pre-processor.

Simply put, a pre-processor is a program that produces output that serves as input to another program. In the present context, the input is a series of SAS statements, and the output is a modified or augmented version of these statements, which is passed to another SAS session for execution (Figure 5). The specific modifications required include:

- Assigning non-standard input (raw) data and derived data libraries (i.e. to run acute-phase programs on extension-phase raw data)
- Pointing to the correct 'vintage' of format catalog and autocall library
- Sub-setting (i.e. to retain only acute-phase records on extension-phase raw data)

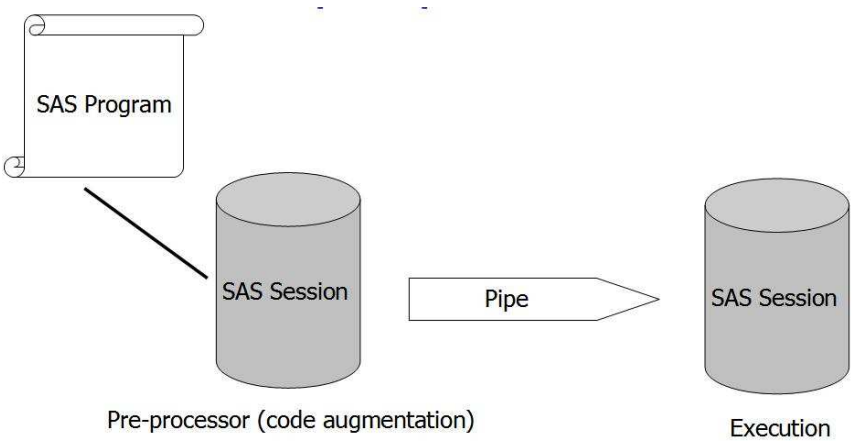

Figure 5. A SAS pre-processor

This pre-processer is implemented using a 'pipe' (an 'interprocess channel'), where the 'standard output' ('stdout', the statements typically written to a console or command-line window) of one SAS session is used as 'standard input' to a second SAS session. The pipe is implemented in a filename statement:

```
filename test_ pipe 'sas9 -sysin /dev/stdin';

data _null_;
  infile '<directory>/myprogram.sas' LRECL=2000 truncover;
  file test_;

  input str_ $2000.;
  put str_;
```

The ‘pre-processor’ SAS session reads the program file ‘myprogram.sas’, and outputs each line of code as ‘standard output’ via the ‘put’ statement, where it is then received as ‘standard input’ by the ‘execution’ SAS session. As you can see, this provides the opportunity in the ‘data _null_’ step for identifying specific lines of code, removing lines of code (i.e. by conditional execution of the ‘put’ statement), or modifying or inserting lines of code. To illustrate, recall that one of the requirements in implementing cross-comparison was to assign non-standard raw-data and derived-data libraries, and this is implemented in this instance simply by writing out extra ‘libname’ statements before the first line of input code is passed to the execution session:

```
if _n_=1 then do;
  str__='libname rawdata "'||"&libraw"||'";';
  put str__;
  str__='libname derdata "'||"&libder"||'";';
  put str__;
end;
```

Note however that the specific placement of these library reassignments depends entirely on how your programming environment is set up (if, for example, you use a ‘setup’ macro called at the top of the program to assign these library references, the pre-processor will instead need to identify the macro call, and place the library re-assignments immediately after it). Identifying specific lines of code can be done via simple text matching, and for the purposes described here this is sufficient; however, by setting things up to use regular expression functions, you can potentially expand the scope of your code augmentation but writing text-substitution rules tailored to your specific requirements. For instance, if you wanted to insert a particular line of code each time a certain type of statement appeared, you might use something like this:

```
pattern = "&regex";
patternID = prxparse(pattern);
call prxsubstr(patternID,str_,position,length);

if position>0 then do;

  put "&insert";

end;
```

In this case, &regex holds the regular expression describing the string (or string type) to identify, and when the string is located, &insert provides the additional code statement to insert (this might be used, for example, to apply subject-level or visit-level sub-setting each time a dataset is read in from the raw-data library).

An example pre-processor macro is provided in Appendix 1; while this macro meets the relatively straight-forward requirements of the cross-comparison described in this paper, some additional functionality is also provided for interest’s sake. With the intent of making search/replace rules ‘expandable’, the macro allows you to store search/replace string pairs in a pipe-delimited text file:

```
expression1 | code statement 1
expression2 | code statement 2
```

Each line of SAS code read by the pre-processor can then be matched against the full suite of expressions stored in this text file, and specific code augmentation applied where a match is found:

```
%let rc=%sysfunc(filename(filrf,"xsubst.txt"));
%let fid=%sysfunc(fopen(&filrf));

  %if &fid > 0 %then %do;

    %do %while(%qsysfunc(fread(&fid)) = 0);

      %let rc=%qsysfunc(fget(&fid,c,200));
      %let c_=%superq(c);

      patternID = prxparse("%substr(&c_,1,%index(&c_,|)-1)");
      call prxsubstr(patternID,STR_,position,length);
```

```
        if position>0 then do;

          put "%substr(&c_,%index(&c_,|)+1)";

        end;
```

In this case, expression/code statement pairs are stored in the file ‘xsubst.txt’, and each SAS statement read by the pre-processor is matched against each expression (again using regular expression functions). Though it is not shown here, it would be a simple matter to control the relative placement of original and inserted code statements in the stream of statements being piped to the execution session (in other words, to place the added statement before or after the original statement, or to replace the original statement altogether).

Back to our cross-comparison implementation, to illustrate the basic components, we start with a derivation program as it stood at the time of the extension-phase lock and analysis:

```
*************************************************************
*************************************************************
**-PROGRAM NAME: D1_DEMOG.SAS
**-
**-DESCRIPTION:  CREATES THE DEMOG ANALYSIS DATA SET.
**-
**-
**-
*************************************************************
*************************************************************;

%include '../tools/automacs.sas';
```

In our programming environment, the raw-data (initdata) and derived-data (sasdata) libraries are assigned via a shell script (which, incidentally, is itself a pre-processor), and the ‘%include’d ‘automacs.sas’ program sets the autocall library reference and format catalog reference. If we want to run this program on archived acute-lock raw data, we need to re-assign the data libraries, and call the ‘automacs.sas’ file that resides in the extension-lock parent directory. When we run this program using the pre-processor, the log shows these modifications:

```
2          libname initdata "/pub/studies/<project>/acute_final/primary/initdata";
NOTE: Libref INITDATA was successfully assigned as follows:
      Engine:        V9
      Physical Name: /pub/studies/<project>/acute_final/primary/initdata
3          libname sasdata "../sasdata/xt_on_au";
NOTE: Libref SASDATA was successfully assigned as follows:
      Engine:        V9
      Physical Name: /pub/studies/<project>/acute_compare/sasdata/xt_on_au
4
*************************************************************************
5
*************************************************************************
6          **-PROGRAM NAME: D1_DEMOG.SAS
7          **-
8          **-DESCRIPTION:  CREATES THE DEMOG ANALYSIS DATA SET.
9          **-
33         **-
34         **-
35
*************************************************************************
36
*************************************************************************;
37
38         %include '/pub/studies/<project>/primary/tools/automacs.sas';
39         libname library '/pub/studies/<project>/primary/formats';
```

Libname statements have been inserted to re-assign the raw-data and derived-data libraries, and the automacs '%include' statement has been identified and replaced with the correct automacs call and format library reference, so the pre-processor has successfully implemented the basic cross-comparison requirements, saving us from either making code modifications by hand, or moving archived data and/or programs around.

The 'test' DEMOG analysis dataset that is written to the '/pub/studies/<project>/acute_compare/sasdata/xt_on_au' library (so named since the dataset is created by running extension-phase code on acute-phase raw data) can now be compared against the original acute-lock DEMOG dataset, and differences between the two can be attributed to changes to program logic.

CONCLUSION

For projects where separate database lock and analysis events takes place at discrete project milestones, we need to be able to document changes that have taken place in both raw data and program logic between these milestones, and to assess the implications of these changes. 'Cross-comparison' provides an effective mechanism for distinguishing between analysis dataset differences resulting from changes to raw data and those resulting from changes to program logic. However, additional work may still be required in order to deal with record-count differences and to detail value-level differences.

Because cross-comparison may be 'retro-fitted' to a project where the requirement was not anticipated in advance, a 'pre-processor' can help in implementation, effectively removing the need to modify legacy code.

The task of documenting 'data drift' is fairly open-ended: while it is relatively easy to document differences, it's a much bigger task to explain them. Where different 'versions' of a database produce differences in statistical inference, it's these inferential differences that may guide the detailed investigation of specific dataset and variables.

REFERENCES

Fairfield-Carter, Brian. 2010. "Insert Diagnostic Code Using Pipes and PRX (Regular Expression) Functions". *Proceedings of the 2010 PharmaSUG Conference.* Available at http://www.lexjansen.com/pharmasug/2010/cc/cc11.pdf.

ACKNOWLEDGMENTS

We would like to thank ICON Clinical Research for consistently encouraging and supporting conference participation, and all our friends at ICON for their great ideas, enthusiasm and support, in particular Syamala Schoemperlen and Jackie Lane. Special thanks to Stephen Hunt for coming up with the concept (and inviting us to steal it :), and to Tony Pisegna for his excellent review comments.

CONTACT INFORMATION

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

Name: Brian Fairfield-Carter, Jasmin Fredette
Enterprise: ICON Clinical Research, Inc.
City, State ZIP: Redwood City, CA
E-mail: fairfieldcarterbrian@gmail.com

APPENDIX 1. PRE-PROCESSOR MACRO

```
%macro preproc(progfile=,
               sdir=,
               libinit=,
               libsas=,
               regex=,
               insert=,
               include_orig=YES);

    filename test_ pipe 'sas9 -sysin /dev/stdin';

    data _null_;
      attrib str_ str__ length=$2000.;
      infile "&progfile" LRECL=2000 truncover;
      file test_;

      input str_ $2000.;

      %*** STUDY-SPECIFIC STUFF THAT NEEDS TO BE ADJUSTED...;

      if _n_=1 then do;
        str__='%let STUDYDIR='||"&sdir"||';';
        put str__;
        str__='libname initdata "'||"&libinit"||'";';
        put str__;
        str__='libname sasdata "'||"&libsas"||'";';
        put str__;
      end;

      %*** USE PRX FUNCTIONS TO IDENTIFY TARGET STRINGS,
            AND INSERT ADDITIONAL CODE;

      pattern = "&regex";
      patternID = prxparse(pattern);
      call prxsubstr(patternID,STR_,position,length);

      if position>0 then do;

        %if &include_orig=YES %then %do;

          put str_; %*** PASS THROUGH THE ORIGINAL SOURCE CODE...;

        %end;

        str__='*** -----INSERTED CODE----- ***;';
        put str__;

        put "&insert";

        str__='*** -----END INSERTED CODE----- ***;';
        put str__;

      end;
      else do;
        put str_; %*** PASS THROUGH THE ORIGINAL SOURCE CODE...;
      end;


      %*** ADDITIONAL TARGET STRINGS AND AUGMENTATION STATEMENTS STORED
           IN A PIPE-DELIMITED TEXT FILE...;
      %*** (WILL ONLY RUN IF THE FILE XSUBSTR.TXT EXISTS, AND WILL RUN
            ONCE FOR EACH LINE FROM THE ORIGINAL SOURCE FILE
```

```
             (NOT SURE WHAT PERFORMANCE WILL BE LIKE WITH LARGE SOURCE FILES OR
              WITH A LARGE NUMBER OF ADDITIONAL TARGET STRINGS));

        %let filrf=myfile;
        %let rc=%sysfunc(filename(filrf,"xsubst.txt"));
        %let fid=%sysfunc(fopen(&filrf));

        %if &fid > 0 %then %do;

          %do %while(%qsysfunc(fread(&fid)) = 0);

            %let rc=%qsysfunc(fget(&fid,c,200));
            %let c_=%superq(c);

            patternID = prxparse("%substr(&c_,1,%index(&c_,|)-1)");
            call prxsubstr(patternID,STR_,position,length);

            if position>0 then do;

              put "%substr(&c_,%index(&c_,|)+1)"; %*** INSERT ADDITIONAL STATEMENT;

            end;

          %end;

        %end;
        %let rc=%sysfunc(fclose(&fid));
        %let rc=%sysfunc(filename(filrf));

     run;

  %mend preproc;


  %preproc(progfile=%str(../analprog/d1_visit.sas),
           sdir=%str(/pub/studies/<sponsor>/<study>/primary),
           libinit=%str(/pub/studies/<sponsor>/<study>/acute_final/primary/initdata),
           libsas=%str(../sasdata/xt_on_au),
           regex=%str(/automacs.sas/i),
           insert=%str(%include
     '/pub/studies/<sponsor>/<study>/primary/tools/automacs.sas';
      libname library '/pub/studies/<sponsor>/<study>/primary/formats';),
           include_orig=NO);
```