SAS® GLOBAL FORUM 2018

USERS PROGRAM

How a Code-Checking Algorithm Can Prevent Errors

April 8 – 11 | Denver, CO

#SASGF

How a Code-Checking Algorithm Can Prevent Errors

Thomas Hirsch

Magellan Health Inc

ABSTRACT

WHEN A COMPANY USES AN AUTOMATED PRODUCTION SYSTEM FOR REPORTING, THERE ARE ALWAYS RISKS OF HAVING RECURRING ERRORS DUE TO ISSUES WITH REPORTS BEING SUBMITTED INCORRECTLY. ONE WAY TO REDUCE THESE ERRORS IS TO UTILIZE A CODE CHECKING PROGRAM WHICH WILL ASSESS SEVERAL ASPECTS OF A PROGRAM BEFORE IT IS SCHEDULED, INCLUDING ITS COMPATIBILITY WITH THE PRODUCTION ENVIRONMENT, INCLUSION OF COMMENTS, AND NOTIFICATION OF SECURITY RISKS. IN THIS PAPER, I WILL BE DISCUSSING SOME OF THE METHODS THAT CAN BE INCLUDED IN A CODE CHECKING PROGRAM, AS WELL AS SOME METHODS TO IMPLEMENT THESE TECHNIQUES. THE FIRST AND MOST IMPORTANT WILL BE SIMULATING A RUN IN AN AUTOMATED PRODUCTION ENVIRONMENT. WE WILL THEN LOOK AT ANALYZING THE VOLUME AND COMPLETENESS OF COMMENTS IN THE CODE BEING TESTED. ALSO, WE WILL REVIEW METHODS TO HANDLE WARNINGS AND OTHER NON-CRITICAL ISSUES THAT COULD BE IDENTIFIED. FINALLY, WE WILL LOOK AT METHODS OF CHECKING FOR RISKY FIELDS BEING USED, INCLUDING PERSONAL OR FINANCIAL INFORMATION WHICH NEED TO HAVE A LIMITED DISTRIBUTION.

INTRODUCTION

Production errors in report services are a drag on your company, costing time, effort, and sometimes even money through fees or penalties. Every company has to have a system in place to ensure that bad code doesn't make it into production and cause these problems. When used in conjunction with good coding standards and proper peer review, a code checking algorythm can further reduce the chance that mistakes can affect standard business procedures. While a code checker can be remarkably flexible, this paper will focus on its ability to test a program's compatibility with the company's automation system, to review the completeness of comments, and ensure high-risk variables are reviewed before they can be seen by the wrong people. After reading this, you should be able to take the provided framework, and adjust it to your own systems and the needs of your industry.

WHAT IS A CODE CHECKER

A Code checker is an automated program that will review code set up for review and identify key points for review before the code is put in production. Generally, you will use your company automation system to run the code checker, either on a set frequency, or checking for when a code has been made available for review, depending on the limits of the system. Once it is active, the code below is used to identify and pull in whatever code is being reviewed by the checker:

WRAPPER FILE

```
options symbolgen;
%let myfilename=RunMe;
Filename filelist pipe "dir /b /s \\phobos\idg\CodeCheck\&myfilename.\*.sas";

Data progs;
        infile filelist truncover;
        input path $100.;
        filename = scan(path,-1,"\");
run;

/*Code Checker Code*/
%include '\\phobos\idg\prod\code\TidalTest\CodeCheckerMacros.sas';e;
%let hdrfile= %str(\\phobos\idg\prod\code\TidalTest\StandardHeaderTemplate.sas);

%macro convert(path, filename, checkstep);
%if &checkstep = 1 %then %do;
%let inpgm = &path.;
        %CC_Initialize;
        %ParseTemplate(hdrfile=&hdrfile);
        %ExamineSASPgm(inpgm=&inpgm,outds=Results);
        %CheckHeader(inds=Results);
        %CheckOther(inds=Results);
        %CheckRisk(inds=Results);
        %CheckComments(inds=Results);
        %ReportOut;
%end;
```

How a Code-Checking Algorithm Can Prevent Errors

Thomas Hirsch

Magellan Health Inc

WRAPPER (Continued)

```
%if &checkstep = 2 %then %do;
options noxwait;
x start/w "" "D:\Program Files\SASHome\SASFoundation\9.4\sas.exe" -sysin &path. -log
"\\phobos\idg\CodeCheck\Completed" -config "\\phobos\idg\prod\code\TidalTest\sasv9_test.cfg" -print
"\\phobos\idg\CodeCheck\Completed" -work "E:\SAS Temporary Files\tidaltest";
x move "&path." "\\phobos\idg\CodeCheck\Completed\&filename.";

data _null_;
logname = tranwrd("&filename.",'sas','log');
call symput("logname",logname);
run;

/*Check log for errors and send completion emails for the job*/
data checklog;
infile "\\phobos\idg\CodeCheck\Completed\&logname." truncover;
input rows $5000.;
ROWS = TRANSLATE(ROWS,' ',"'", " ",'"');
IF SUBSTR(ROWS,1,5)='ERROR:' OR SUBSTR(ROWS,1,7)='WARNING:'
OR INDEX(UPCASE(ROWS),"UNINITIALIZED") > 0
OR INDEX(UPCASE(ROWS),"_ERROR_") > 0
OR INDEX(UPCASE(ROWS),"REPEATS OF BY VALUES") > 0
OR INDEX(UPCASE(ROWS),"EXTRANEOUS") > 0
OR INDEX(UPCASE(ROWS),"INVALID DATA FOR") > 0
OR INDEX(UPCASE(ROWS),"SAS SYSTEM STOPPED PROCESSING") > 0
OR INDEX(UPCASE(ROWS),"INVALID ARGUMENT") > 0
OR INDEX(UPCASE(ROWS),"ODS PDF PRINTED NO OUTPUT") THEN OUTPUT;
run;
```

```
filename mymail email to              = ("TPHirsch@magellanhealth.com");
subject = "&filename completed test run";

data _null_;
file mymail;
set checklog;
if _n_ = 1 then put "Log is accessible at \\phobos\idg\CodeCheck\Completed\&logname."
                // " If this meets peer review approval, please attach the log to the JIRA ticket."
                // "File &filename generated the following warnings and errors:" //;
if _n_ ge 1 then put @4 rows //;
run;
%end;

%mend;

/*Step one - Code Check*/;
data _null_;
                set progs;
call execute('%nrstr(%convert('||path||','||filename||',1))');
run;

/*Step two - Code Execution*/;
data _null_;
                set progs;
call execute('%nrstr(%convert('||path||','||filename||',2))');
run;
```

How a Code-Checking Algorithm Can Prevent Errors

Thomas Hirsch

Magellan Health Inc

RUNNING THE CODE IN THE SYSTEM

Probably the most important step to prevent errors is to make sure that the program runs in the production environment. If, as was recommended in the step above, you have set up a recurring process in the production environment for this code checker, it is a simple process from here to run the file. You can manually start a SAS job which will run the code in question, and utilize the same rules as your production environment.

Let's break down the elements of this code.

- We are starting a sas process, using the SAS program, which should be updated to your environment.
- The Sysin command tells the SAS session to immediately run the code in question when it opens
- Log, config, print, and work define where we want these test logs and elements to be saved. Config in particular should be a file that is updated to ensure that this is reflective of production rules.

RUNNING TEST CODE

```
x start/w "" "C:\Program Files\SASHome\SASFoundation\9.4\sas.exe"
-sysin &path.
-log "C:\CodeCheck\Completed"
-config "C:\code\TidalTest\sasv9_test.cfg"
-print "C:\CodeCheck\Completed"
-work "C:\SAS Temporary Files\tidaltest"; x start/w "" "C:\Program Files\SASHome\SASFoundation\9.4\sas.exe"
-sysin &path.
-log "C:\CodeCheck\Completed"
-config "C:\code\TidalTest\sasv9_test.cfg"
-print "C:\CodeCheck\Completed"
-work "C:\SAS Temporary Files\tidaltest";
```

RUNNING ERROR CHECK

After running the above code, you can add additional elements as needed. One reccomendation is the below code, which can be used to parse the log for errors, warnings, and other elements that should be noted by the developer.

ERROR CHECK CODE

```
data _null_;
logname = tranwrd("&filename.",'sas','log');
call symput("logname",logname);
run;

/*Check log for errors and send completion emails for the job*/
data checklog;
infile "C:\CodeCheck\Completed\&logname." truncover;
input rows $5000.;
ROWS = TRANSLATE(ROWS,' ','"', " ","'");
IF SUBSTR(ROWS,1,5)='ERROR:' OR SUBSTR(ROWS,1,7)='WARNING:'
OR INDEX(UPCASE(ROWS),"UNINITIALIZED") > 0
OR INDEX(UPCASE(ROWS),"_ERROR_") > 0
OR INDEX(UPCASE(ROWS),"REPEATS OF BY VALUES") > 0
OR INDEX(UPCASE(ROWS),"EXTRANEOUS") > 0
OR INDEX(UPCASE(ROWS),"INVALID DATA FOR") > 0
OR INDEX(UPCASE(ROWS),"SAS SYSTEM STOPPED PROCESSING") > 0
OR INDEX(UPCASE(ROWS),"INVALID ARGUMENT") > 0
OR INDEX(UPCASE(ROWS),"ODS PDF PRINTED NO OUTPUT") THEN OUTPUT;
run;
```

How a Code-Checking Algorithm Can Prevent Errors

Thomas Hirsch

Magellan Health Inc

REVIEWING COMMENTS

It is important for code to have sufficient documentation, especially when you have a large team that may have to take on one another's work at a moment's notice. There are a few ways that can be monitored. Ones we will be looking at below are header checks and comment density.

HEADER FILE ANALYSIS

Most quality code will have a header at the top. This will include basic information like code name, frequency, source tables, etc. The below code will scan the header portion of the document, and check for key items, and verify if they have been filled out.

COMMENT COUNT

In addition to checking the header, we can also review each step of code and determine how much of it has commenting. While this is by no means a fool-proof check, it can at the very least serve as a warning if the developer sees that a large number of their statements are lacking comments

HEADER ANALYSIS CODE

```
%let hdrfile= %str(C:\prod\code\TidalTest\StandardHeaderTemplate.sas);
data test;

        set results;

        if _n_ > 1 then stop;

        length ErrorMsg $200;

        array Keyword {20} $ 50;
        array aType {20} $ 1;
        array aLen {20} 8;
```

HEADER CODE (Continued)

```
        ErrorType = 'CHKHEADER';

        *** Load Keywords from Standard Header Template ***;
        do i = 1 to 100;
                set Keywords end=last;
                Keyword{i} = Key;
                aType{i}   = Type;
                aLen{i}                 = Len;
                if last then do;
                        ikey = i;
                        i = 101;
                end;
        end;

        *** Compress Statement ***;
        Statement = compress(Statement,'*','s');

        *** Length Validation ***;
        if length(statement) >= 4000 then do;
                ErrorMsg = 'ERROR: Standard Header Too Long';
                ERROR ErrorMsg;
                rc=dosubl('%InsertError('||ErrorType||', '||ErrorMsg||');');
                stop;
        end;

        WorkStatement = Statement;
        LenStatement = length(Statement);
```

How a Code-Checking Algorithm Can Prevent Errors

Thomas Hirsch

Magellan Health Inc

HEADER CODE (Continued)

```
do i = 1 to ikey;
        CKW = keyword{i};
        IsCKWFound      = index(WorkStatement,trim(CKW));
        LenCKW      = length(CKW);

        if i<ikey then do;      NKW = keyword{i+1};         end;
        else do;                NKW = 'HIDDENKEYWORD:';     end;

        IsNKWFound = index(WorkStatement,trim(NKW));
        IsNKWFound = ifn(IsNKWFound = 0,LenStatement,IsNKWFound);

        if IsCKWFound > 0 then do;
          NKWExpPos         = IsCKWFound+LenCKW+aLen{i};
          if NKWExpPos > isNKWFound then do;
                ErrorMsg =
'WARNING: No Value Found for Keyword: '|| Keyword{i};
                ERROR ErrorMsg;
                rc=dosubl(
'%InsertError('||ErrorType||', '||ErrorMsg||');');
           end;
        end;
        else do;
         ErrorMsg = 'WARNING: Missing Keyword in Header: '|| Keyword{i};
         ERROR ErrorMsg;
         rc=dosubl('%InsertError('||ErrorType||', '||ErrorMsg||');');
        end;
end;
run;
```

COMMENT COUNT CODE

```
data _null_;
        set &inds end=eof;
        retain CommentCount  StepCount 0;
        ErrorType = 'INFO';

        if _n_ = 1 then do;
                PrevStepNum = StepNum;
                PrevStepName = StepName;
        end;
        else do;
                PrevStepNum = lag(StepNum);
                PrevStepName = lag(StepName);
                ** Increment StepCount only for DATA and PROC **;
                if          PrevStepNum ne StepNum
                        and (StepName = 'DATA' or StepName = 'PROC')
                                    then StepCount = StepCount + 1;
                ** Increment Comment Count **;
                if          PrevStepName ne StepName
                        and PrevStepName = 'COMMENT'
                                    then CommentCount = CommentCount + 1;
                if eof then do;
ErrorMsg = 'INFO: 9002 '||compbl(put(CommentCount,5.)||'out of '||put(StepCount,5.)||' steps had comments');
                  putlog ErrorMsg;
                  rc=dosubl('%InsertError('||ErrorType||', '||ErrorMsg||');');
                end;
        end;
run;
```

How a Code-Checking Algorithm Can Prevent Errors

Thomas Hirsch

Magellan Health Inc

CHECKING FOR HIGH RISK FIELDS

For every company, there are certain elements that are risky to release in reports. Social Security Numbers, Credit Card numbers, or any other personal information can be a risk on any report. While there are always exceptions that will need this information, you can eliminate a lot of risk by having an automated system that will let you know when these high-risk elements are included in release code.

FIELD ANALYSIS CODE

```
data testout;
        set results;

        length ErrorMsg $200;

        ErrorType = 'DATA RISK';

        *** Compress Statement ***;
        Statement = compress(compress(statement,,'kw'));

        *** COB Sum Fix;
if upcase(StepName) ne 'COMMENT' and
index(upcase(Statement),'I_OTHER_PAYER_AMT') > 0 then do;
if index(upcase(Statement),'sum(I_OTHER_PAYER_AMT)') > 0 then do; end;
                else do;
                        ErrorMsg = 'ERROR: A1 COB Other
Payer included without COB Sum Fix';
                        ERROR ErrorMsg;
                        rc=dosubl('%InsertError('||ErrorType||', '||ErrorMsg||');'
);
                end;
        end;
```

HEADER CODE (Continued)

```
        *** New Financial Fields;
        if upcase(StepName) ne 'COMMENT' and
        (index(upcase(Statement),'O_TOTAL_AMT_PAID') > 0
or index(upcase(Statement),'O_DISPENSE_FEE_PAID_AMT') > 0
          or index(upcase(Statement),'O_INGRED_COST_PAID_AMT')>0) then do;
                if index(upcase(Statement),'O_TOTAL_AMT_PAID') > 0 then
                        ErrorMsg = 'WARNING: A2 Using Financial Fields with
Internal Data: O_TOTAL_AMT_PAID, replace
TOTAL_CLIENT_AMT_BILLED';
                if index(upcase(Statement),'O_DISPENSE_FEE_PAID_AMT') > 0 then
                        ErrorMsg = 'WARNING: A2 Using Financial Fields with
Internal Data: O_DISPENSE_FEE_PAID_AMT, replace with
REPRICE_DISP_FEE_AMT';
                if index(upcase(Statement),'O_INGRED_COST_PAID_AMT') > 0 then
                        ErrorMsg = 'WARNING: A2 Using Financial Fields with
Internal Data: O_INGRED_COST_PAID_AMT, replace with
REPRICE_INGRED_AMT';
                        ERROR ErrorMsg;
                        rc=dosubl('%InsertError('||ErrorType||', '||ErrorMsg||');');
        end;
run;
```

How a Code-Checking Algorithm Can Prevent Errors

Thomas Hirsch

Magellan Health Inc

SAMPLE EMAILS

Log is accessible at [redacted]

If this meets peer review approval, please attach the log to the JIRA ticket.

File drug_trend_new_8.sas generated the following warnings and errors:

pricing_con=$-0.01 (-49.96%) inflation_con=$0.01 (20.2%) drug_mix_con=$0 (0%) _ERROR_=1 _N_=14

pricing_con=$0 (.%) inflation_con=$0 (.%) drug_mix_con=$0 (.%) _ERROR_=1 _N_=96

inflation_con=$0.01 (20.2%) drug_mix_con=$0 (0%) _ERROR_=1 _N_=143

cost_share_con=$0 (.%) pricing_con=$0 (.%) inflation_con=$0 (.%) drug_mix_con=$0 (.%) _ERROR_=1

pricing_con=$0 (.%) inflation_con=$0 (.%) drug_mix_con=$0 (.%) _ERROR_=1 _N_=1299

Code Checker Results:
Filename : [redacted]

CHKHEADER :
WARNING: No Value Found for Keyword: INPUT:
WARNING: Missing Keyword in Header: INCLUDECALLS:
WARNING: Missing Keyword in Header: EXTERNALFILECALL:

DATA RISK :
WARNING: A2 Using Financial Fields with Internal Data: O_TOTAL_AMT_PAID
WARNING: A2 Using Financial Fields with Internal Data: O_TOTAL_AMT_PAID
WARNING: A2 Using Financial Fields with Internal Data: O_TOTAL_AMT_PAID

INFO :
INFO: 9002 26 out of 238 steps had comments

Thanks
Business Intelligence Team

CONCLUSIONS

A Code Checker can be a way to improve productivity and save time with errors and production issues. While it is important to ensure that you have customized the system to your own situation, this framework is flexible enough that it can be a boon to whatever your environment looks like.

SAS® GLOBAL FORUM 2018

April 8 – 11 | Denver, CO
Colorado Convention Center

#SASGF

Paper 2798-2018

How a Code-Checking Algorithm Can Prevent Errors

Thomas Hirsch, Magellan Health Inc.

ABSTRACT

When a company uses an automated production system for reporting, there are always risks of having recurring errors due to issues with reports being submitted incorrectly. One way to reduce these errors is to utilize a code checking program which will assess several aspects of a program before it is scheduled, including its compatibility with the production environment, inclusion of comments, and notification of security risks. In this paper, I will be discussing some of the methods that can be included in a code checking program, as well as some methods to implement these techniques. The first and most important will be simulating a run in an automated production environment. We will then look at analyzing the volume and completeness of comments in the code being tested. Also, we will review methods to handle warnings and other non-critical issues that could be identified. Finally, we will look at methods of checking for risky fields being used, including personal or financial Information which need to have a limited distribution.

INTRODUCTION

Production errors in report services are a drag on your company, costing time, effort, and sometimes even money through fees or penalties. Every company has to have a system in place to ensure that bad code doesn't make it into production and cause these problems. When used in conjunction with good coding standards and proper peer review, a code checking algorythm can further reduce the chance that mistakes can affect standard business procedures. While a code checker can be remarkably flexible, this paper will focus on its ability to test a program's compatibility with the company's automation system, to review the completeness of comments, and ensure high-risk variables are reviewed before they can be seen by the wrong people. After reading this, you should be able to take the provided framework, and adjust it to your own systems and the needs of your industry.

WHAT IS A CODE CHECKER, AND HOW DOES IT WORK?

A Code checker is an automated program that will review code set up for review and identify key points for review before the code is put in production. Generally, you will use your company automation system to run the code checker, either on a set frequency, or checking for when a code has been made available for review, depending on the limits of the system. Once it is active, the code below is used to identify and pull in whatever code is being reviewed by the checker:

```
%macro setfilename();
%let myfilename=RunMe;

Filename filelist pipe "dir /b /s C:\&myfilename.\*.sas";

Data progs;
    infile filelist truncover;
    input path $100.;
    filename = scan(path,-1,"\");
run;
%mend;

%setfilename();

/*Code Checker Code*/
%include 'C:\TidalTest\CodeCheckerMacros.sas';
```

After the checker has identified the code being reviewed, it will serve as a wrapper file, pulling in whatever macros have been identified by your team as critical for analysis. Some of these macros will be discussed in later sections.

The final step for the code checker will be to provide the results of the check. This can be done through an automated e-mail. For our team, we send this message team-wide, so the review process can be collaborative as needed. See below for a sample of the output code we have used:

```
filename mymail email to      = ("<email_address_here>");
subject = "&filename completed test run";

data _null_;
file mymail;
set checklog;

put "Log is accessible at C:\CodeCheck\Completed\&logname."
      // " If this meets peer review approval, please attach the log to
the JIRA ticket."
      // "File &filename generated the following warnings and errors:"
//;

run;
```

MACROS TO IDENTIFY CODING RISKS

While the code checker is active, you can use different macros to identify key potential problems in code scheduled for production. Below we will provide some common and effective macros that can be used or modified as needed. Below is some sample code for integrating these macros into the overall code checker:

```
%let myfilename=RunMe;

%let inpgm = &path.;
    %CC_Initialize;
    %ParseTemplate(hdrfile=&hdrfile);
    %ExamineSASPgm(inpgm=&inpgm,outds=Results);

    %CheckHeader(inds=Results);
    %CheckOther(inds=Results);
    %CheckRisk(inds=Results);
    %CheckComments(inds=Results);
    %ReportOut;
```

In the above sample, path is the file location, while the first three macros help to break out the code into segments. They will be included in the appendix for details. The remaining macros are the individual elements that can be added as needed to provide additional checks on code to be published.

RUNNING THE CODE IN THE AUTOMATED SYSTEM

Probably the most important step to prevent errors is to make sure that the program runs in the production environment. If, as was recommended in the step above, you have set up a recurring process in the production environment for this code checker, it is a simple process from here to run the file. You can manually start a SAS job which will run the code in question, and utilize the same rules as your production environment. We will use the code below as an example:

```
x start/w "" "C:\Program Files\SASHome\SASFoundation\9.4\sas.exe"
    -sysin &path.
    -log "C:\CodeCheck\Completed"
```

```
-config "C:\code\TidalTest\sasv9_test.cfg"
-print "C:\CodeCheck\Completed"
-work "C:\SAS Temporary Files\tidaltest";
```

Let's break down the elements of this code.

- We are starting a sas process, using the SAS program, which should be updated to your environment.
- The Sysin command tells the SAS session to immediately run the code in question when it opens
- Log, config, print, and work define where we want these test logs and elements to be saved. Config in particular should be a file that is updated to ensure that this is reflective of production rules.

After running the above code, you can add additional elements as needed. One reccomendation is the below code, which can be used to parse the log for errors, warnings, and other elements that should be noted by the developer:

```
data _null_;
logname = tranwrd("&filename.",'sas','log');
call symput("logname",logname);
run;

/*Check log for errors and send completion emails for the job*/
data checklog;
infile "C:\CodeCheck\Completed\&logname." truncover;
input rows $5000.;
ROWS = TRANSLATE(ROWS,' ','"', " ","'");
IF SUBSTR(ROWS,1,5)='ERROR:' OR SUBSTR(ROWS,1,7)='WARNING:'
OR INDEX(UPCASE(ROWS),"UNINITIALIZED") > 0
OR INDEX(UPCASE(ROWS),"_ERROR_") > 0
OR INDEX(UPCASE(ROWS),"REPEATS OF BY VALUES") > 0
OR INDEX(UPCASE(ROWS),"EXTRANEOUS") > 0
OR INDEX(UPCASE(ROWS),"INVALID DATA FOR") > 0
OR INDEX(UPCASE(ROWS),"SAS SYSTEM STOPPED PROCESSING") > 0
OR INDEX(UPCASE(ROWS),"INVALID ARGUMENT") > 0
OR INDEX(UPCASE(ROWS),"ODS PDF PRINTED NO OUTPUT") THEN OUTPUT;
run;
```

The final element I would recommend in this portion is to have its own e-mail separate from the other elements, since the log will likely have its own issues which should be viewed seperately from the other warnings:

```
filename mymail email to      = ("<email_address_here>");
subject = "&filename completed test run";

data _null_;
file mymail;
set checklog;

put @4 rows //;

run;
```

REVIEWING COMMENTS WITHIN CODE FOR COMPLETENESS

It is important for code to have sufficient documentation, especially when you have a large team that may have to take on one another's work at a moment's notice. There are a few ways that can be monitored. Ones we will be looking at below are header checks and comment density.

Assessing Header Quality

Most quality code will have a header at the top. This will include basic information like code name, frequency, source tables, etc. The below code will scan the header portion of the document, and check for key items, and verify if they have been filled out:

```
%let hdrfile    = %str(C:\prod\code\TidalTest\StandardHeaderTemplate.sas);
data test;

    set results;

    if _n_ > 1 then stop;

    length ErrorMsg $200;

    array Keyword {20} $ 50;
    array aType {20} $ 1;
    array aLen {20} 8;

    retain ikey 1;

    ErrorType = 'CHKHEADER';

    *** Load Keywords from Standard Header Template ***;
    do i = 1 to 100;
          set Keywords end=last;
          Keyword{i}  = Key;
          aType{i}    = Type;
          aLen{i}           = Len;
          if last then do;
                ikey = i;
                i = 101;
          end;
    end;

    *** Compress Statement ***;
    Statement = compress(Statement,'*','s');

    *** Length Validation ***;
    if length(statement) >= 4000 then do;
          ErrorMsg = 'ERROR: Standard Header Too Long';
          ERROR ErrorMsg;
          rc=dosubl('%InsertError('||ErrorType||', '||ErrorMsg||');');
          stop;
    end;

    WorkStatement = Statement;
    LenStatement = length(Statement);

    do i = 1 to ikey;

          CKW = keyword{i};
          IsCKWFound  = index(WorkStatement,trim(CKW));
```

```
        LenCKW      = length(CKW);

        if i<ikey then do;
              NKW   = keyword{i+1};
        end;
        else do;
              NKW = 'HIDDENKEYWORD:';
        end;

        IsNKWFound = index(WorkStatement,trim(NKW));
        IsNKWFound = ifn(IsNKWFound = 0,LenStatement,IsNKWFound);

        if IsCKWFound > 0 then do;
           NKWExpPos      = IsCKWFound+LenCKW+aLen{i};
           if NKWExpPos > isNKWFound then do;
              ErrorMsg =
                    'WARNING: No Value Found for Keyword: '|| Keyword{i};
              ERROR ErrorMsg;
              rc=dosubl(
                      '%InsertError('||ErrorType||', '||ErrorMsg||');');
           end;
        end;
        else do;
          ErrorMsg = 'WARNING: Missing Keyword in Header: '|| Keyword{i};
          ERROR ErrorMsg;
          rc=dosubl('%InsertError('||ErrorType||', '||ErrorMsg||');');
        end;
   end;
   run;
```

Counting Code Included for Each Step

In addition to checking the header, we can also review each step of code and determine how much of it has commenting. While this is by no means a fool-proof check, it can at the very least serve as a warning if the developer sees that a large number of their statements are lacking comments:

```
   data _null_;
      set &inds end=eof;

      retain CommentCount     StepCount 0;

      ErrorType = 'INFO';

      if _n_ = 1 then do;
            PrevStepNum = StepNum;
            PrevStepName = StepName;
      end;
      else do;
            PrevStepNum = lag(StepNum);
            PrevStepName = lag(StepName);


            ** Increment StepCount only for DATA and PROC **;
            if          PrevStepNum ne StepNum
                  and (StepName = 'DATA' or StepName = 'PROC')
                        then StepCount = StepCount + 1;
```

```
            ** Increment Comment Count **;
            if          PrevStepName ne StepName
                  and PrevStepName = 'COMMENT'
                        then CommentCount = CommentCount + 1;

            if eof then do;
               ErrorMsg = 'INFO: 9002 '||compbl(put(CommentCount,5.)||' out
                        of '||put(StepCount,5.)||' steps had comments');
               putlog ErrorMsg;
               rc=dosubl('%InsertError('||ErrorType||', '||ErrorMsg||');');
            end;
      end;
  run;
```

CHECKING FOR HIGH-RISK FIELDS

For every company, there are certain elements that are risky to release in reports. Social Security Numbers, Credit Card numbers, or any other personal information can be a risk on any report. While there are always exceptions that will need this information, you can eliminate a lot of risk by having an automated system that will let you know when these high-risk elements are included in release code.

```
data testout;
      set results;


      length ErrorMsg $200;


      ErrorType = 'DATA RISK';


      *** Compress Statement ***;
      Statement = compress(compress(statement,,'kw'));


      *** COB Sum Fix;
      if upcase(StepName) ne 'COMMENT' and
                  index(upcase(Statement),'I_OTHER_PAYER_AMT') > 0 then do;
            if index(upcase(Statement),'sum(I_OTHER_PAYER_AMT)') > 0 then do;
            end;
            else do;
                  ErrorMsg = 'ERROR: A1 COB Other
                                    Payer included without COB Sum Fix';
                  ERROR ErrorMsg;
                  rc=dosubl('%InsertError('||ErrorType||', '||ErrorMsg||');'
                        );
            end;
      end;
```

```
      *** New Financial Fields;
      if upcase(StepName) ne 'COMMENT' and
      (index(upcase(Statement),'O_TOTAL_AMT_PAID') > 0
         or index(upcase(Statement),'O_DISPENSE_FEE_PAID_AMT') > 0
         or index(upcase(Statement),'O_INGRED_COST_PAID_AMT')>0) then do;
            if index(upcase(Statement),'O_TOTAL_AMT_PAID') > 0 then
                  ErrorMsg = 'WARNING: A2 Using Financial Fields with
                        Internal Data: O_TOTAL_AMT_PAID, replace
                        TOTAL_CLIENT_AMT_BILLED';
            if index(upcase(Statement),'O_DISPENSE_FEE_PAID_AMT') > 0 then
                  ErrorMsg = 'WARNING: A2 Using Financial Fields with
                        Internal Data: O_DISPENSE_FEE_PAID_AMT, replace with
                        REPRICE_DISP_FEE_AMT';
            if index(upcase(Statement),'O_INGRED_COST_PAID_AMT') > 0 then
                  ErrorMsg = 'WARNING: A2 Using Financial Fields with
                        Internal Data: O_INGRED_COST_PAID_AMT, replace with
                        REPRICE_INGRED_AMT';
            ERROR ErrorMsg;
            rc=dosubl('%InsertError('||ErrorType||', '||ErrorMsg||');');
      end;
run;
```

CONCLUSION

A Code Checker can be a way to improve productivity and save time with errors and production issues. While it is important to ensure that you have customized the system to your own situation, this framework is flexible enough that it can be a boon to whatever your environment looks like.

APPENDIX

Macros used in standard practices:

CC_Initialize

```
  ** Delete Error Dataset if exists **;
  %if %sysfunc(exist(Error)) ne 0 %then %do;
      proc datasets noprint; delete Error; run;
      %put *** Error Dataset Deleted ***;
  %end;

  ** Load Steps **;
  proc sql noprint;
      create table StepName as
      (     select      *
            from  Steps
      );
  quit;
  %let nStep = &sqlobs;
```

```
  %let CC_Initialize = 1;

ParseTemplate
  %if &hdrfile = %str() %then %do;
      %put ERROR: Standar Header Template was not specified;
      %goto MacroEnd;
  %end;

  %if %sysfunc(fileexist(&hdrfile)) %then %do;
      filename hdrfile "&hdrfile";
  %end;
  %else %do;
      %put ERROR: Standard Header Template does not exist;
      %goto MacroEnd;
  %end;

  data Keywords(keep=Key Type Len);

      length statement $4096;
      length textn $200;

      array Keyword {100} $ 50;
      array aType {100} $ 1;
      array aLen {100} 8;

      retain statement;
      retain ikey 1;
      retain Keyword;

      IsComplete = 0;

      infile hdrfile truncover filename = tmp end=eof;
            * reading of the SAS code as a text file;
      input textn $char201.;
            * the whole line is treated as one character variable;

      textn = compress(textn,,'s');

      if textn = '' then delete;

      IsColonFound = index(textn,':');
      if IsColonFound > 0 then do;
            Keyword{ikey} = compress(substr(textn,1,IsColonFound));
            if index(textn,':N/A') then do;
                  aType{ikey} = 'O';
                  aLen{ikey} = 0;
            end;
            else do;
                  aType{ikey} = 'M';
                  aLen{ikey} = length(compress(textn,'?','K'));
            end;

            Key = Keyword{ikey};
            Type = aType{ikey};
            Len = aLen{ikey};
            output Keywords;
```

```
            ikey = ikey+1;
        end;

        if length(statement) >= 3896 then do;
            putlog 'ERROR: Standard Header Too Long';
            stop;
        end;

        endpos = index(textn,';');

        if endpos = 0 then do;
            statement = cats(statement,textn);
        end;
        else do;
            statement = cats(statement,substr(textn,1,endpos));
            IsComplete = 1;
        end;

        if IsComplete then do;
            stop;
        end;
    run;
```

ExamineSASPgm

```
    %if &CC_Initialize ne 1 %then %do;
        %put ERROR: Please include/run CodeChecker Config process;
        %goto MacroEnd;
    %end;

    %if &inpgm = %str() %then %do;
        %put ERROR: Input Program Name not found;
        %goto MacroEnd;
    %end;

    %if &outds = %str() %then %do;
        %put ERROR: Output Dataset not found - Defaulted to Statements;
        %let outds = Statements;
    %end;

    %if %sysfunc(fileexist(&inpgm)) %then %do;
        filename myfiles "&inpgm";
    %end;
    %else %do;
        %put ERROR: Input Program does not exist;
        %goto MacroEnd;
    %end;

    %let lg= 200; *declare the length of each input line;

    /* Identifies Steps and Statements */
    DATA  results(keep=StepName StepNum Statement StmtNum);

        *length environment $4 ffolder $15 tmp fname $80;
        length statement $4096;
        length textn $&lg.; * defining the length of the new string;
        length CurrStep $30;
```

```
    array Step {&nStep} $ 30      Step1 - Step%eval(&nStep);

    infile myfiles truncover filename = tmp end=eof;
          * reading of the SAS code as a text file;

    input textn $char%eval(&lg+1).;
          * the whole line is treated as one character variable;

    textn = compress(textn,,'c');

    * Initialize all flags;
    if _n_ = 1 then do;
          SQuote            = 0;
          DQuote            = 0;
          SComment    = 0;
          DComment    = 0;
          Statement   = '';
          CurrStep    = '';
          StepNum     = 0;
          StmtNum     = 0;
          LineNo = 0;
    end;

    LineNo = LineNo + 1;

    * Flags that carry informatio across lines need to be retained;
    retain      Statement   SQuote      DQuote      SComment    DComment
    CurrStep    StepNum     StmtNum     LineNo;

    * Initialize counters for every line;
    i = 1;
    endpos = 0;

    * Load StepNames;
    do i = 1 to &nStep;
          set StepName point=i;
          Step{i} = StepStart;
          *put Step{i};
    end;

    /* Parse thru the input line and write Statements*/
    i = 0;
    do while (i < length(textn));

          i = i+1;

          OneChar = substr(textn,i,1);
          TwoChar = substr(textn,i,2);

*put OneChar '/' TwoChar '//' SComment '/' DComment '//' SQuote '/' DQuote;

    if SComment = 0 and DComment = 0 then do;
          if OneChar= "'" then if SQuote= 0 then SQuote= 1; else SQuote= 0;
          if OneChar= '"' then if DQuote= 0 then DQuote= 1; else DQuote= 0;
    end;

    if OneChar = '*' and TwoChar ne '*/' and SQuote = 0 and DQuote = 0 and
```

```
      SComment = 0 then SComment = 1;
if OneChar = ';' and SComment = 1 then SComment = 0;

if DComment = 0 then if TwoChar = "/*"
      then do; DComment = 1; i=i+1; end;
if DComment = 1 then if TwoChar = '*/'
      then do; DComment = 0; i=i+1; end;

if SComment = 0 and DComment = 0 and SQuote = 0 and DQuote = 0 then do;
      if (  OneChar = ';' or
                  TwoChar = '*/'          )
      then do;
            endpos = i;
            * Write Out Statement;
            StmtNum = StmtNum + 1;
            statement = cats(statement,substr(textn,1,i),'\n');
            *put Statement;
            * Examine the Statement to identify steps;
            FirstWord = upcase(scan(statement,1));
            StepName = CurrStep;
            put FirstWord;
            if FirstWord = 'RUN' or FirstWord = 'QUIT' then do;
                  StepNum = StepNum + 1;
                  StepName = CurrStep;
                  CurrStep = '';
            end;
            else if substr(trim(statement),1,1) = '%'
               and FirstWord = 'INCLUDE' then do;
                  StepNum = StepNum + 1;
                  StepName = 'INCLUDE';
                  CurrStep = '';
            end;
            else if substr(trim(statement),1,1) = '%' then do;
                  StepNum = StepNum + 1;
                  StepName = 'MACRO';
                  CurrStep = '';
            end;
            else if substr(statement,1,1) = '*' or
              substr(statement,1,2) = '/*' then do;
                  if CurrStep = 'DATA' or CurrStep = 'PROC' then do;
                        StepNum = StepNum + 1;
                        StepName = 'DATACOMMENT';
                  end;
                  else do;
                        StepNum = StepNum + 1;
                        StepName = 'COMMENT';
                        CurrStep = '';
                  end;
            end;
            else do;
                  do s = 1 to &nStep;
                  if index(FirstWord,scan(Step{s},1)) > 0
                     and CurrStep = '' then do;
                              StepNum = StepNum + 1;
                              StepName = scan(Step{s},1);
                              if scan(Step{s},2) = '1' then do;
                                    CurrStep = StepName;
```

```
                                    end;
                                    else
                                          CurrStep = '';
                                    leave;
                              end;
                              else do;

                              end;
                        end;
                  end;
                  output results;
                  statement = '';
            end;
      end;

      end;

      if i > endpos+1 then do;
            statement = cats(statement,substr(textn,endpos+1,i-endpos));
      end;

      if eof then do;
            if lengthn(trim(statement)) > 0 then do;
                  output results;
                  put 'Incomplete Last Statement Found';
            end;
      end;
run;
```

CONTACT INFORMATION

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

Thomas Hirsch
Magellan Health Inc
Isaic16@gmail.com