Paper 150-31

Exploiting Consistent Program Documentation

Richard Koopmann Jr, Capella University, Minneapolis, MN

ABSTRACT

Just because documenting programs isn't fun, doesn't mean it shouldn't be done. You've probably had to work off programs written several years before by someone who wasn't exactly faithful about program documentation. This makes for a headache when trying to figure out what the point of a program is and why it was coded the way it was. Fast-forward a few years...will your predecessors be frustrated by your lacking documentation, or will they be grateful that you spent the time to describe the program? Perhaps programmers feel that by not documenting programs they have more job-security, or maybe it's form of hazing; either way, it didn't help you. How will it help your predecessors?

This paper moves beyond the simple practice of documenting programs to developing and implementing a system of automatic documentation. The results, in native SAS® data set, can then be put through various PROCs to practically any format to look like you spent hours drudging through your programs.

While the collection of steps described in this paper were developed using SAS 9.1.3 originally running under Windows 2000 and then Windows XP, the basic concepts (and hopefully much of the code) should be transferable to other versions of SAS and other operating systems.

INTRODUCTION

So you've been documenting your programs, right? Have you been doing it consistently? If the answer of either of these questions is 'No', perhaps this paper can help change your programming habits by revealing some potential benefits. Before reading Frank DiIorio's paper on building utility macros and programs, my answers were 'Yes' and 'Not really' to these questions. While I made a good effort a documenting the programs that I thought needed to be documented (which was typically limited to the complex ones), I certainly wasn't doing it consistently across programs. Sometimes I'd add the date the program was created, and sometimes not, sometimes I'd document changes, and sometimes not. Sound familiar?

That all changed when I discovered abbreviations and keyboard macros. Abbreviations expand text via keyboard macros. I set up an abbreviation to expand `/**/` to a standard program header (see Code Listing 1 for an example) that I use in all programs that take me more than 10 minutes to code. I've also assigned shortcut key combinations to certain keyboard macros. If I'm feeling nimble, I can use the assigned key combo `Ctrl+Alt+1` to insert the program header; similarly, `Ctrl+Alt+2` inserts a section marker to make browsing long code easier. So abbreviations, keyboard macros, and shortcut key combinations were a quick way to insert consistent text.

After viewing the code listings scattered throughout this paper, you may be curious why I end block comments with a semicolon; Block comments outside of data and proc steps end up getting folded when you collapse a step in the enhanced editor. I find this very annoying, especially when your paging through a tall program. With a semicolon at the end of block comments, the enhanced editor seems to treat the comment block as a separate step and will not fold these parts into neighboring steps.

CONSISTENCY BETWEEN PROGRAMS

The best thing any programmer can do is to develop and adopt a set of programming conventions and stick with them. Too often, I'd be faced with programs that used different naming conventions (some used `onewordnames` or `CamelCaseNames` others used `underscored_names`). For example, sometimes the unique record identifier was coded as `ID`, sometimes as `uniqueid` or `UniqueID`, and occasionally as `Unique_ID`. When merging data sets, inconsistencies become headaches. I'd end up going into older data sets and to standardize things.

DOCUMENT PROGRAMS CONSISTENTLY

Just using the same conventions between programs is a good start, but if you are not consistently documenting your programs, you're only doing half the job. If you think a program is straight forward enough that documentation is not necessary, you may kick yourself three months down the road when you've got to go back and modify it to add new functionality. A good question to ask yourself is 'Would I mind re-writing this program in a month?'. If your answer is no, you should document it.

REAPING THE HARVEST OF YOUR CONSISTENCY

Assuming you've set your documenting conventions, stuck to them between programs, and have diligently documented a number of your programs, you're ready take advantage of your consistency. Figure 1 shows the process overview.

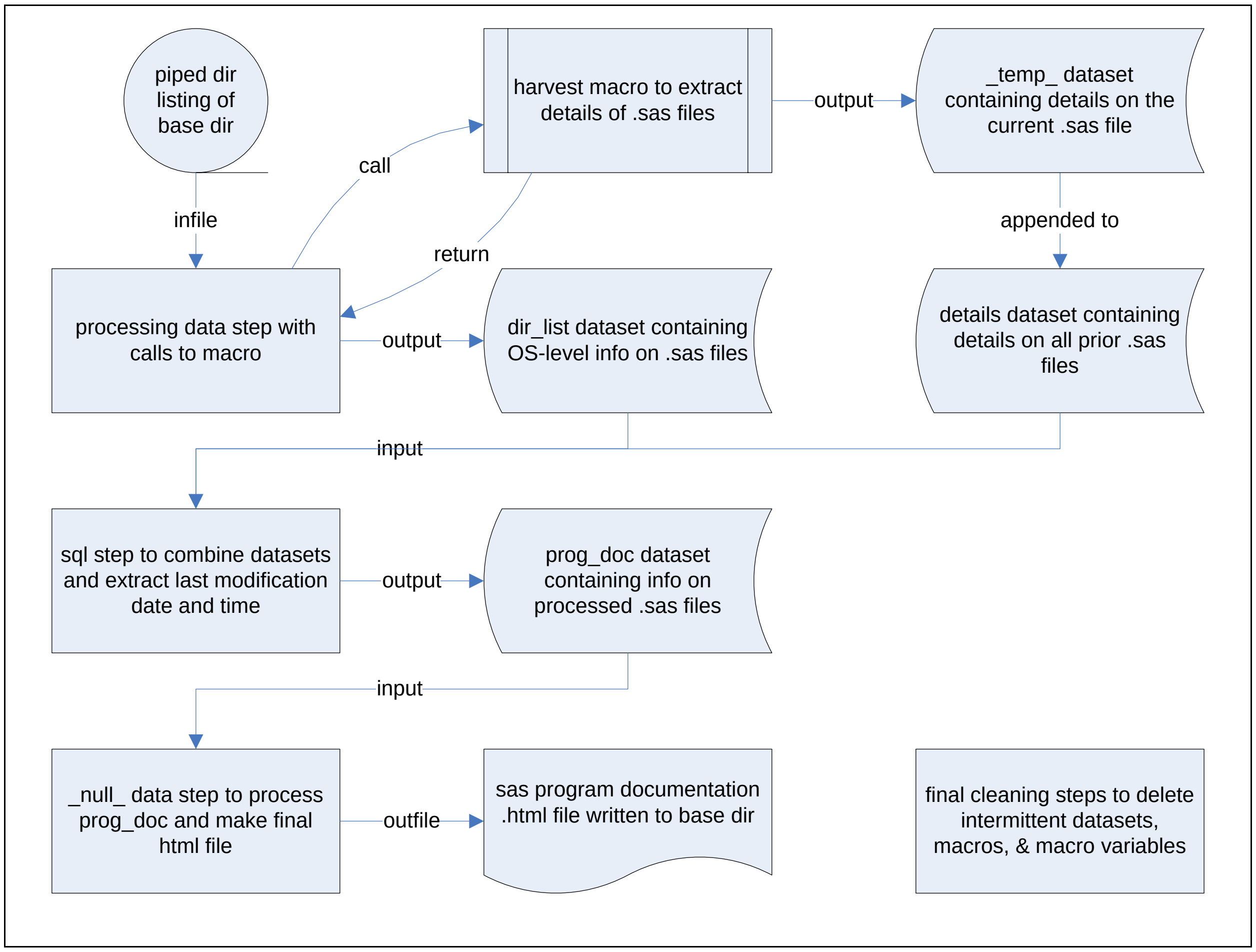

piped dir listing of base dir
infile
call
harvest macro to extract details of .sas files
output
temp dataset containing details on the current .sas file
appended to
return
processing data step with calls to macro
output
dir_list dataset containing OS-level info on .sas files
details dataset containing details on all prior .sas files
input
sql step to combine datasets and extract last modification date and time
output
prog_doc dataset containing info on processed .sas files
input
null data step to process prog_doc and make final html file
outfile
sas program documentation .html file written to base dir
final cleaning steps to delete intermittent datasets, macros, & macro variables

Figure 1. Process Overview.

THE STANDARD PROGRAM HEADER LAYOUT

Basic assumption of the collection of steps in this paper is that the program headers are in several of your programs. Different programmers will inevitably find different parts more or less important and should add and drop sections from the headers they adopt. The important thing is to consistently use a standard program header. An example of the standard program header layout Harvest macro expects can be seen in Code Listing 1; note that the macro prohibits page breaks within elements.

```
/*
  PROGRAM INFORMATION
    PROJ: SUGI31 Poster
    DESC: Provides aggregated program documentation.
    AUTH: Richard Koopmann richard.koopmann@capella.edu
    DATE: 2005-08-11

  INPUTS/OUTPUTS
    INs : A base directory.
    OUTs: HTML file in base directory.
```

```
     MODIFICATIONS
       2006-01-19 Richard Koopmann richard.koopmann@capella.edu
         (+) Added section headers.
       2005-09-21 Richard Koopmann richard.koopmann@capella.edu
         (+) Added a proc datasets to clean up the dropping.
         (+) Added a data null step to generate html report.
       2005-08-31 Richard Koopmann richard.koopmann@capella.edu
         (+) Added code line number.
         (+) Added view combining file listing and detail listing.
       2005-08-30 Richard Koopmann richard.koopmann@capella.edu
         (+) Accommodate arguments of macro programs.
       2005-08-26 Richard Koopmann richard.koopmann@capella.edu
         (+) Filelist dataset includes only true .sas files.
       2005-08-25 Richard Koopmann richard.koopmann@capella.edu
         (+) Added Prog_Doc dataset.
         (-) Removed SQL proc to get paths and filenames.
         (*) Fixed Filesize field.
   */;
```

Code Listing 1. The Standard Program Header.

THE PIPED DIRECTORY LISTING

A piped fileref with the appropriate command switches (`/s /-c` in Windows XP) is used to retrieve a recursive dir listing from the specified base directory; note the switch to suppress the thousand separator.

Also note the format of the date and time format of a directory listing can vary depending on your localized settings and on the operating system. For example, Windows 2000 reports 03:00p while Windows XP reports 03:00 PM. This is a subtle but important difference since the Dir_List Data Step uses SAS regular expressions and functions to sift through the piped directory entries.

```
/*********************************************************************************\
THE PIPED DIRECTORY LISTING
\*********************************************************************************/;
%let basedir=\\mspfile03\users\rkoopmann\SAS\projects;
filename dirlist pipe "dir &basedir.\*.* /s /-c";

%let dirdate=\d\d\/\d\d\/\d\d\d\d;
%let dirtime=\d\d:\d\d [A|P]M;
```

Code Listing 2. The Piped Directory Listing.

THE HARVEST MACRO

Since we will be reading an unknown number of files spread across an unknown number of subdirectories, a macro is the most efficient way of shuffling through the input files. The Harvest macro only needs to know the name and location of the input file when it is called from within the Dir_List data step. One option to speed up the documentation processing by only reading the program header portion of the input files (everything between `/*` and `*/;`). Since I did not intended on running this program very often, this option was not coded. I have set up a flag variable to use null input statements (`input;`) once the input line position moves out of the program header. This allows for a total line count for each program.

The Harvest macro contains a `clear` flag that, when set to 1 will remove any artifacts (if they exist) from prior runs. If these artifacts are not cleaned up, new information would be appended to old information. This ensures the program is starting from a clean slate when processing the directory listing.

```
/*********************************************************************************\
THE HARVEST MACRO
\*********************************************************************************/;
```

```
%macro harvest(path,name,clear);
  %if &clear %then %do;
    proc datasets library=work nolist;
      delete details _temp_;
    quit;
  %end;

  data _temp_(drop=buff isHead);
    format buff $250.   path $200.   name $30.   line 8.   field $15.   value $100.;
    retain isHead   path "&path."   name "&name."   field;

    infile "&path.\&name." EOF=LastRec missover dsd;
    input  buff $ @;

    if _N_=1 & substr(buff,1,2) = '/*' then isHead=1;
    line = _N_;

    if isHead then do;
      select (upcase(substr(buff,1,4)));
        when ('PROJ') do; field='Project';       value=substr(buff,7);    end;
        when ('DESC') do; field='Description';   value=substr(buff,7);    end;
        when ('AUTH') do; field='Author';        value=substr(buff,7);    end;
        when ('DATE') do; field='Create Date';   value=substr(buff,7,10); end;
        when ('INS' ) do; field='Inputs';        value=substr(buff,7);    end;
        when ('OUTS') do; field='Outputs';       value=substr(buff,7);    end;
        when ('REQS') do; field='Required Args'; value=substr(buff,7);    end;
        when ('OPTS') do; field='Optional Args'; value=substr(buff,7);    end;
        when ('2005','2006') do;  /* should be a better method */
          field='Modified on'; value=substr(buff,1,10); output;
          field='Modified by'; value=substr(buff,12);
          end;
        when ('(+)','(-)','(*)') do;
          select (substr(buff,2,1));
            when ('+') field='Added';
            when ('-') field='Removed';
            when ('*') field='Changes';
            otherwise;
            end;
          value=substr(buff,5);
          end;
        when ('/*','PROG','INPU','MODI','ARGU') delete;
        when ('*/','*/;') isHead=0;
        otherwise do;
          if trim(buff)='' then delete;
          else value=buff;
          end;
        end;
      output;
      end;
    else input;
    return;
    LastRec:
    line=7;
    field='Total Lines';
    value=_n_;
    output;
  run;

  proc datasets library=work nolist;
```

```
    append base=details data=_temp_;
  quit;
%mend harvest;
```

Code Listing 3. The Harvest Macro.

THE DIRECTORY LISTING DATA STEP

The Directory Listing Data Step is a fairly simple and straightforward (elegant when compared to the Harvest macro) data step converts the piped directory listing into a dataset containing operating system-level information (e.g., path, filename, size, and a date-time stamp of when the program was last modified). At the end of every step, the program will call the Harvest macro; on the first call to the macro, the `clear` flag is set to trigger pre-cleaning. Since the macro implements an appending datasets procedure, all the details get dumped into the same SAS dataset (dir_list).

```
/*********************************************************************************\
THE DIRECTORY LISTING DATA STEP
\*********************************************************************************/;
data dir_list;
  if _N_ = 1 then do;
    retain re_file re_dir re_dirname firstfile;
    re_file=prxparse("/(&dirdate.)\s+(&dirtime.)\s+(\d+)\s+(\S.*)/");
    re_dir =prxparse("/(&dirdate.)\s+(&dirtime.)\s+<DIR>\s+(\S.*)/");
    re_dirname=prxparse("/Directory of (.+)/");
    firstfile=1;
    end;

  length dirname name $ 128;
  format DateTime dateampm16.;
  keep   dirname datetime size name;
  retain dirname;

  infile dirlist;
  input;

  if lengthn(_infile_); * ignore blank records;

  * Parse file records;
  if prxmatch(re_file, _infile_) then do;
    call prxposn(re_file,1,pos,len); date=input(substr(_infile_,pos,len),mmddyy10.);
    call prxposn(re_file,2,pos,len); time=input(substr(_infile_,pos,len),time6.);
    call prxposn(re_file,3,pos,len); size=input(substr(_infile_,pos,len),comma32.);
    call prxposn(re_file,4,pos,len); name=substr(_infile_,pos,len);
                                     DateTime=dhms(date,0,0,time);
    if upcase(scan(name,-1,'.')) eq "SAS" then
          call execute('%harvest(' || trim(dirname) || ','
                                   || trim(name)    || ',' || firstfile || ')');
    if firstfile then firstfile=0;
    output;
    end;

  * Parse directory records;
  else if prxmatch(re_dir, _infile_) then do;
    call prxposn(re_dir,1,pos,len); date=input(substr(_infile_,pos,len),mmddyy10.);
    call prxposn(re_dir,2,pos,len); time=input(substr(_infile_,pos,len),time6.);
    call prxposn(re_dir,3,pos,len); name=substr(_infile_,pos,len);
                                    DateTime=dhms(date,0,0,time);
    if name ~ in ('.','..') then output;
    end;
  * parse base dir records;
  else if prxmatch(re_dirname,_infile_) then do;
    call prxposn(re_dirname,1,pos,len); dirname=substr(_infile_,pos,len);
    end;
run;
```

Code Listing 4. The Directory Listing Data Step.

THE PROGRAM DOCUMENTATION DATASET

The Program Documentation dataset contains all the information required to make the Program Documentation HTML File. Information such as line count, author, description, project and modification entries can be found here. The SQL proc also extracts the date and time of the last update to the dataset used in the processing data step below.

```
  /*********************************************************************************\
  THE PROGRAM DOCUMENTATION DATASET
  \*********************************************************************************/;
  proc sql noprint;
    *merge directory dataset with prog_doc dataset;
    create table Prog_Doc as
    select distinct dir.dirname, dir.name, dir.size, det.line, det.field, det.value
    from dir_list as dir left join details as det
      on dir.dirname = det.path and dir.name = det.name
    order by dir.dirname, dir.name, det.line
    ;

    *get 'last modified' datetime;
    select datepart(modate) format=worddate., timepart(modate) format=timeampm8.
    into :mdate, :mtime
    from dictionary.tables
    where libname eq "WORK" and memname eq "PROG_DOC"
    ;
  quit;
```

Code Listing 5. The Program Documentation Dataset.

THE PROGRAM DOCUMENTATION PROCESSING DATA STEP

This data step steps through the Program Documentation Data Set and writes the Program Documentation HTML File along with the JavaScript and cascading style sheets (CSS) needed for enhanced functionality. Pointer controls are used to write the familiar nested indentation that many programmers use. Note the JavaScript and CSS sections are hard-coded in the file; If you will be using this for several independent directories of programs, you should consider linking to external .js and .css files.

```
  /*********************************************************************************\
  The SAS Program Documentation.html Generating _null_ Data Step
  \*********************************************************************************/;
  data _null_;
    set Prog_Doc(where=(Field is not null)) end=isEnd;
    file "&basedir.\SAS Program Documentation.html";
    by dirname name;

    * variables used throughout the program;
    retain sq "'" Proj ProjName Prog ProgName;
    sval=compress(value,' /');
    snam=compress(name, ' /');
    dir=substr(dirname,28);
```

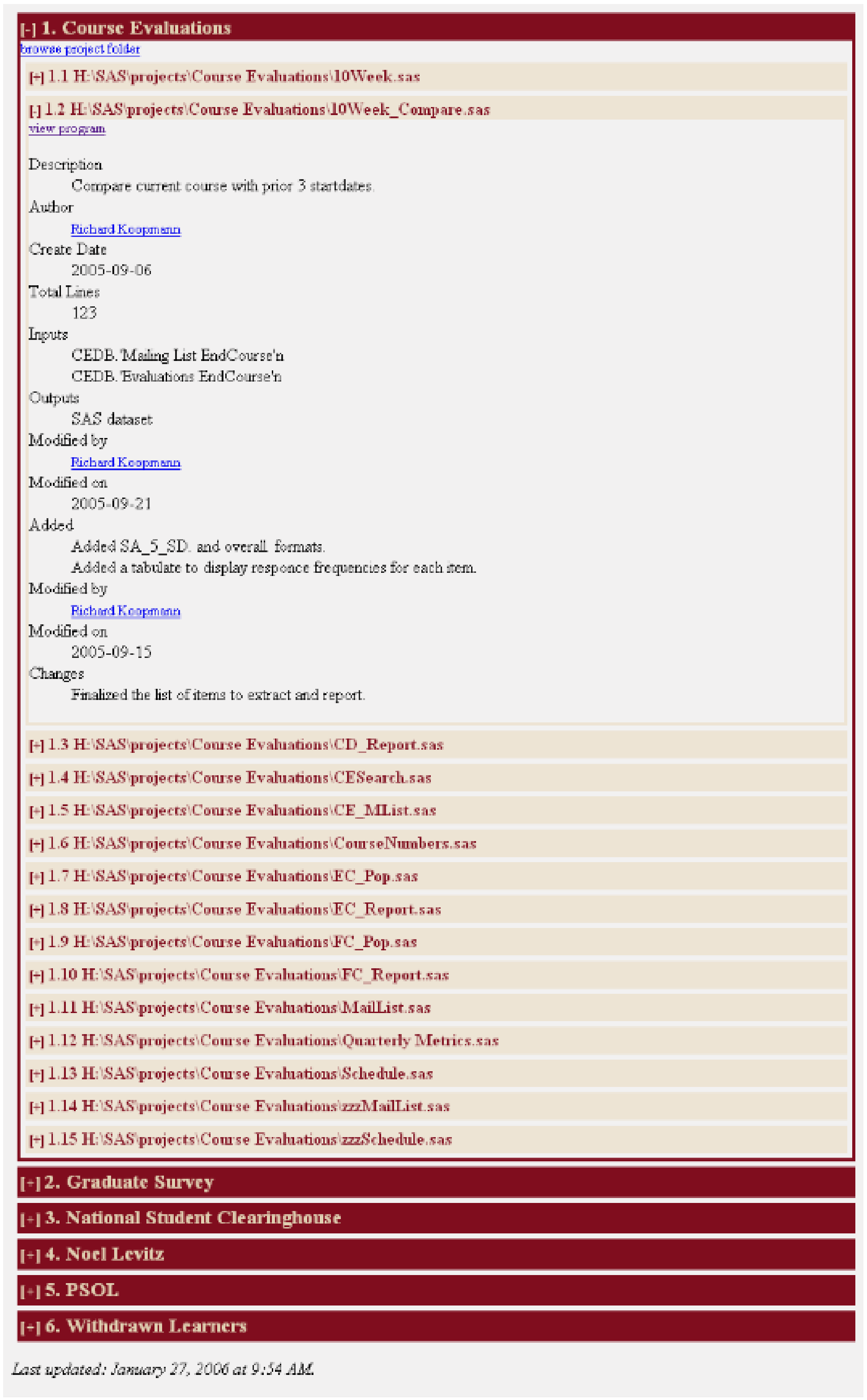
```
*Initialize HTML file;
if _n_=1 then do;
  put @01 '<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">'
     /@01 '<html>'
     /@03 '<head>'
     /@05 '<title>Program Documentation</title>'
     /@05 '<script language="Javascript">'
     /@07 'function toggle( targetId ) {'
     /@09 'if ( document.getElementById ) {'
     /@11 'target = document.getElementById( targetId );'
     /@11 'label  = document.getElementById( "label" + targetId );'
     /@11 'if ( target.className == "isClosed" )    // if currently isClosed'
     /@11 '   { target.className =  "isOpened";     // set to isOpened'
     /@11 '     label.className  =  "setClosed"; }'
     /@11 'else'
     /@11 '   { target.className = "isClosed";      // else, hide'
     /@11 '     label.className = "setOpened"; }'
     /@09 '}'
     /@07 '}'
     /@05 '</script>'
    //@05 '<style type="text/css">'
     /@07 'body { background: #F2F1F1; width: 50em; }'
     /@07 'h1 { font-size: 120%; font: bolder; color: #E0D0B2; '
          'background-color: #800D1E; cursor: pointer; margin: 0;}'
     /@07 'h2 { font-size: 100%; font: bolder; color: #800D1E; '
          'background-color: #EDE4D4; cursor: pointer; margin: 0;}'
     /@07 'a { font-size: 80%; }'
     /@07 'div.project { border: solid #800D1E; margin: 5px; padding: 0; }'
     /@07 'div.program { border: solid #EDE4D4; margin: 5px; padding: 0; }'
     /@07 '.isOpened { display: block; }'
     /@07 '.isClosed { display: none;  }'
     /@07 '.setOpened::before { content: "[+] "; font-size: 80%; }'
     /@07 '.setClosed::before { content: "[-] "; font-size: 80%; }'
     /@05 '</style>'
     /@03 '</head>'
    //@03 '<body>'
      ;
  end;

* Start of project;
if first.dirname then do;
  Proj+1; ProjName = value;
  Prog=.;
  put/@05 '<div class="project">'
     /@07 '<h1 class="setClosed" id="label' sval
           +(-1) '" onClick="javascript:toggle(' sq +(-1) sval +(-1) sq
           +(-1) ')">' PROJ +(-1) '. ' value +(-1) '</h1>'
     /@07 '<div class="isOpened" id="' sval +(-1) '">';
  end;

* Start of program;
if first.name then do;
  if first.dirname then put @09 '<a href="file:///H:' dir
                                 +(-1) '">browse project folder</a>';
  Prog+1; ProgName = name;
  put/@09 '<div class="program">'
     /@11 '<h2 class="setOpened" id="label' snam
           +(-1) '" onClick="javascript:toggle(' sq +(-1) snam +(-1) sq
           +(-1) ')">' PROJ +(-1) '.' PROG ' H:' dir +(-1) '\' name
```

```
                +(-1) '</h2>';
        if not last.name then; put @11 '<div class="isClosed" id="' snam +(-1) '">';
        if first.path then;    put @13 '<a href="file:///H:' dir +(-1) '\' name
                                        +(-1) '">view program</a>';
        if not last.name then; put @13 '<dl>' @;
        end;

      if not first.name and not last.name then do;
        if field ne 'Project' then do;
          if (lag(field)~=field) then /* Do not repeat change type */
          put @17 '<dt>' field +(-1) '</dt>';

          select (upcase(field));
            when ('AUTHOR','MODIFIED BY') do;
              author=scan(value,1) || ' ' || scan(value,2);
              email=scan(value,-1,' ');
              put @17 '<dd><a href="mailto:' email +(-1) '?subject=' %trim(ProjName)
                       +(-1) ': ' %trim(ProgName) +(-1) '">' author +(-1) '</a></dd>';
              end;
            otherwise do;
              put @17 '<dd>' value +(-1) '</dd>';
              end;
            end;
          end;
        end;

      * End of program;
      if last.name then do;
        if not first.name then; put @13 '</dl>';
        if last.dirname then;   put @11 '</div>';
        put @09 '</div>';
        end;

      * End of project;
      if last.dirname then do;
        put @07 '</div>'
           /@05 '</div>';
        end;

      * Finalize HTML file;
      if isEnd then do;
        put @05 "<p><em>Last updated: &mdate. at &mtime..</em></p>"
           /@03 '</body>'
           /@01 '</html>';
        end;
    run;
```

Code Listing 6. The SAS Program Documentation.html Generating _null_ Data Step.

THE PROGRAM DOCUMENTATION HTML FILE

Figure 2 shows the final product. The html file makes use of simple JavaScript and CSS to display projects and programs which can be expanded and collapsed by pressing the `[+]` and `[-]` symbols preceding the project and program numbers. Projects and programs are numbered in alphabetical order with the first program of the first project being numbered 1.1 (this corresponds to a program called 10week.sas in the Course Evaluation project). Directly below each project and program, there is a link to display the project folder and program code, respectively. Within each program, there are email links (which will populate the subject line with the project name and program name when clicked) to email the author/reviser questions. Lastly, at the bottom of the file, there is a date and time stamp indicating when the file was last updated.



[-] 1. Course Evaluations

browse project folder

[+] 1.1 H:\SAS\projects\Course Evaluations\10Week.sas

[-] 1.2 H:\SAS\projects\Course Evaluations\10Week_Compare.sas

view program

Description
    Compare current course with prior 3 startdates.
Author
    Richard Koopmann
Create Date
    2005-09-06
Total Lines
    123
Inputs
    CEDB.'Mailing List EndCourse'n
    CEDB.'Evaluations EndCourse'n
Outputs
    SAS dataset
Modified by
    Richard Koopmann
Modified on
    2005-09-21
Added
    Added SA_5_SD. and overall. formats.
    Added a tabulate to display responce frequencies for each item.
Modified by
    Richard Koopmann
Modified on
    2005-09-15
Changes
    Finalized the list of items to extract and report.

[+] 1.3 H:\SAS\projects\Course Evaluations\CD_Report.sas

[+] 1.4 H:\SAS\projects\Course Evaluations\CESearch.sas

[+] 1.5 H:\SAS\projects\Course Evaluations\CE_MList.sas

[+] 1.6 H:\SAS\projects\Course Evaluations\CourseNumbers.sas

[+] 1.7 H:\SAS\projects\Course Evaluations\EC_Pop.sas

[+] 1.8 H:\SAS\projects\Course Evaluations\EC_Report.sas

[+] 1.9 H:\SAS\projects\Course Evaluations\FC_Pop.sas

[+] 1.10 H:\SAS\projects\Course Evaluations\FC_Report.sas

[+] 1.11 H:\SAS\projects\Course Evaluations\MailList.sas

[+] 1.12 H:\SAS\projects\Course Evaluations\Quarterly Metrics.sas

[+] 1.13 H:\SAS\projects\Course Evaluations\Schedule.sas

[+] 1.14 H:\SAS\projects\Course Evaluations\zzzMailList.sas

[+] 1.15 H:\SAS\projects\Course Evaluations\zzzSchedule.sas

[+] 2. Graduate Survey

[+] 3. National Student Clearinghouse

[+] 4. Noel Levitz

[+] 5. PSOL

[+] 6. Withdrawn Learners

Last updated: January 27, 2006 at 9:54 AM.

Figure 2. The SAS Program Documentation.html File.

CLEANING UP AFTER YOURSELF

Lastly, we clean up evidence that we were in the work library. Macro variables, filerefs, intermittent datasets, and the compiled harvest macro are all expunged. While the collection of steps described in this paper use the work library exclusively (which is temporary by nature and will be purged by default after your SAS session has ended), it is still good practice to keep your libraries tidy.

```
/*********************************************************************************\
CLEANNING UP AFTER YOURSELF
\*********************************************************************************/;
%let basedir=;
filename dirlist clear;
%let dirdate=;
%let dirtime=;
proc datasets library=work nolist;
  delete _temp_ dir_list details Prog_Doc;
quit;
proc catalog catalog=work.sasmacr entrytype=macro;
  delete harvest;
quit;
```

Code Listing 7. Cleaning Up After Yourself.

POSSIBILITIES

This SAS portion of this program has many areas that can be improved upon including efficiency issues and employing XML tagsets rather than HTML. The Harvest macro would also benefit from the flexibility of regular expressions to identify and categorize lines in program headers rather than relying on exact locations.

The program could be adjusted to examine directories separately and place a file in each directory describing only .sas files found in that directory. Since I wanted central documentation of programs, I opted for the single .html file output in the base directory.

The Program Documentation HTML File can also be improved upon. An interesting feature that DiIorio described is the ability to hyperlink between entries in the modification portion of the program header and the corresponding spot in the program. Since my strict programming habits dictate that I must remove as much clutter from my code as possible without impacting program functionality and readability, modification artifacts are quickly purged. An interesting, and free, syntax highlighter (http://www.dreamprojections.com/SyntaxHighlighter/) could be incorporated to provide a familiar view when browsing the Program Documentation HTML File.

CONCLUSION

I don't know of a programmer who thinks documenting programs is fun. I don't think it is supposed to be. But it is necessary. Hopefully this paper along with the collection of steps will encourage better documentation habit for us programmers.

REFERENCES

DiIorio, Frank. 2005. *Rules for Tools – The SAS Utility Primer* (Paper No. 268-30). http://www2.sas.com/proceedings/sugi30/268-30.pdf

ACKNOWLEDGMENTS

While this program makes use of methods gleaned from many different sources, these were the most important:

- Basic idea of harvesting program headers came from Frank DiIorio's excellent paper *Rules for Tools*.
- JavaScript code to toggle `<div>` tags taken from http://www.drupalart.org/togglediv
- DIR Parsing DATA Step based on SAS Support example http://support.sas.com/rnd/base/topics/datastep/perl_regexp/prxdir.html

RECOMMENDED READING

Aster, Rick. 2000. *Professional SAS Programming Logic.* Paoli, PA: Breakfast Communications Corporation.

Cody, Ron. 2004. *An Introduction to Perl Regular Expressions in SAS 9* (Paper 265-29), http://www2.sas.com/proceedings/sugi29/265-29.pdf

Grant, Paul. *Creating Code Templates in the SAS Enhanced Editor Using Abbreviations and User Defined Keywords.* http://support.sas.com/sassamples/papers/0303_saseditor.pdf

CONTACT INFORMATION

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

Richard Koopmann Jr.
Capella University
Minneapolis, MN 55402
E-mail: richard.koopmann@capella.edu