

Generating a Detailed Table of Contents for Web-Served Output

Derek Morgan, Washington University Medical School, St. Louis, MO

Steve Hoffner, Washington University Medical School, St. Louis, MO

ABSTRACT

ODS HTML is a powerful tool that can automatically generate a hyper-linked table of contents for output produced by the SAS® System. What happens when this excellent functionality doesn't quite go far enough? If you have a multi-user application, you need some way to distinguish between output for user A and user B, even if they have used the same program to produce different results. What happens if the output is directed to different physical web locations? If some of the output is generated outside of the SAS System (for example, with a shell command), ODS won't even know that it exists. How can you generate and maintain a table of contents that will allow users to obtain the specific output they want without having to search through a list of identical contents?

This paper details a macro which generates a unique file name for each piece of output, allows for dynamic generation of a document title, routes the output to multiple web destinations and creates a hyper-linked index page to the output.

WHY?

The previous generations of our SAS/AF and SAS/FSP-based data entry systems¹ all ran on the Windows platform using a single personal computer in each of multiple field centers. In that environment, all hardcopy output could be sent directly to an attached printer using either ODS PRINTER or by issuing an old-fashioned "PRINT" command from CALL SYSTEM (SCL) or an X command within the application.

All of this changed with the current generation, which has been ported to run on a LINUX machine over the internet using an X-window interface. The system became a multi-user system instead of a single-user system, and the application no longer had direct control over the printer. Indeed, some users may not even have printers directly attached to the machines being used to access the application. How could we get the output from our server to paper in their location? Federal Express was not an option because of the on-demand nature of many of the printing tasks, not to mention the expense. Having multiple users access the application created an issue of how do people know which output belongs to them, and which belongs to their co-workers? There is a security concern, too. Having users from multiple field centers accessing the same application and working with the same databases means that you need to route output so that no field center has access to any other field center's output (or data, but that's a topic outside the scope of this paper.) We also wanted to limit the window of access to the output.

Finally, there is also the issue of output that is generated from within the application, but not with ODS. We create barcoded labels using a DATA_NULL_ report, a freeware Code 39 PostScript font, and an in-house macro library². The output is generated as PostScript code. Under Windows, it is easy to send this to an attached PostScript printer, even if that printer does have to be connected to a single user's system via a parallel port. The main reason this in-house library is being used is that many of the barcode generation programs have been in use in other applications for ten years, predating ODS, and thereby removing the necessity of reinventing the wheel. In addition, although it requires programming and trial-and-error layout work, the macro library allows a great deal of control over font sizing and page layout in a straightforward fashion within a DATA step. Since it produces output without using ODS, it won't show up in an ODS-generated table of contents.

THE FILE FORMAT SOLUTION

First and foremost, we needed some way to turn the output into printable form. The PostScript code required the most consideration, since the barcoded labels must print to the same specifications regardless of the printer or software. Ease of access through the web was also a major consideration. And we're running the SAS System on LINUX. Given that combination of factors, any solution involving Microsoft Word was out.

We chose PDF as the solution. If you click on a link from a suitably configured browser, Adobe Acrobat Reader will open and display the document. It's that simple from the end user's standpoint. Even better, it does it for free. How do you convert PostScript code into PDF under LINUX? The "ps2pdf" utility in Ghostscript takes care of that. For free. What about any output generated by SAS System procedures such as PRINT and REPORT? ODS PDF FILE=. That was an easy solution to the first problem. The PDF format takes care of all of our production issues.

THE FILENAME SOLUTION

The next problem was one of generating unique filenames. With multiple users, it is very possible that two users could use the same function in the system to generate different output. For example, two users are printing labels for two different sets of people. How do we make the distinction between the two so that one does not overwrite the other? What about the confidentiality problem, where users from different field centers should not have access to output (and data) that is not their own?

We start this solution with the fact that the data entry system has its own authentication protocol³. At any given point in the application, we can easily obtain who the current user is, and what field centers that user belongs to. We can tag the filename with the user name and use the field center information to direct the output to a specific directory on the web server. A user can generate more than one piece of output per session, so we also need a way to distinguish files between different pieces of output from the same user. We can create a unique filename by appending the datetime value from the system to the user name, since the same user cannot generate two pieces of output at the same time. (If two people are using the same userid and generate output at the same time, they deserve what they get.) Instead of always having to pass this filename to the statement that defines the output filename (which may be in a %INCLUDE file), we can define a symbolic file reference to the file that is to be generated. It is a simple matter to use either the FILENAME statement or the FILENAME() function in SCL to assign the complete file identifier to a symbolic called "OUT". Therefore, we can use "ODS PDF FILE=out;" or "FILE out;" inside of a DATA step in an external program without having to worry about passing filenames.

THE USER INTERFACE SOLUTION

The last problem is to provide a way for users to get the files from a web page. We needed to include hyperlinks and a way for users to identify the output they've generated. We can't use the actual filenames, because names like "derek1375830010.8.pdf" mean absolutely nothing, especially if there are five or six files belonging to "derek". We needed a way to create a text description of the output and give it a hyperlink. This description needed to be dynamically generated, since a single user can perform the same task multiple times for different people, and there needs to be a way of differentiating each piece of output. If we put all of this information into a SAS System table, creating the table of contents web page is easy. ODS HTML and PROC PRINT are painless. There is a directory for each field center, so this HTML file is written to each field center's output directory using a WHERE clause based on field center identifier. That way, the HTML file only contains records for that given field center. The last step is to set up an authentication protocol for the web server, which allows us to restrict access to directories based on user. That takes care of the security problem.

%GETDESC

We created a macro called %GETDESC, which generates file names dynamically, allows for semi-dynamic creation of file descriptions, assigns the FILENAME, maintains the data table, and creates the HTML index file. The macro as used in production is switchable between Windows and LINUX/UNIX. Only the LINUX portion, broken into functional segments, is shown here.

```
1  %MACRO getdesc(desc=,lim=3,ds=tempy);
2  %GLOBAL nomplume pdfout fcidir dirloc;
```

The macro has three parameters (line 1): DESC, which is the descriptive text used in the hyperlink. LIM is the maximum number of individual identifiers to print as a part of the descriptive text, and DS is the name of the SAS System dataset from which the individual identifiers are obtained. If &DS is blank, then individual identifiers will not be added to the description. &NOMPLUME is a macro variable of the filename without any location information or file extension. This is used within the application (and therefore has to be global in scope) wherever it needs an actual file name and a symbolic identifier will not suffice. &PDFOUT is the fully qualified name of the PDF output file, &FCIDIR is the directory name of the field center, and &DIRLOC is the directory name. In production, this is used to account for the difference between LINUX file names ("/") and Windows file names ("").

```
3  DATA _NULL_;
4  LENGTH random_name $ 20;
5  random_name = TRIM(SYMGET('userid')) || LEFT(PUT(DATETIME(),12.1));
6  CALL SYMPUT('nomplume',TRIM(random_name));
7  RUN;
```

The previous segment generates the random name. Using 12 digits for the datetime value will allow the macro to run for another 237 years, give or take a few days.

```
8  %IF &ds<> %THEN %DO;
9  DATA desc;
10 LENGTH output_file dirloc $ 200 filedesc $ 80 link $ 300 fci $ 1
    userid $ 8;
11 RETAIN filedesc;
12 SET &ds END=eof;
13 userid = SYMGET('userid');
14 IF _n_ EQ 1 THEN
15     filedesc = "&desc for ID";
16 IF _n_ LT &lim THEN
17     filedesc = TRIM(LEFT(filedesc)) || ' ' || TRIM(id) || ",";
18 IF eof THEN DO;
19     l = LENGTH(filedesc) - 1;
20     filedesc = SUBSTR(filedesc,l,l);
```

```

21      IF _n_ GT &lim THEN
22          filedesc = TRIM(filedesc) || "... " || " " || TRIM(id);
23          datetime = DATETIME();
24          date = DATEPART(datetime);
25          time = TIMEPART(datetime);
26          fci = SYMGET('fci');
27          CALL SYMPUT('fcidir',TRIM(PUT(fci,$fcdir.)));
28          dirloc = '/users/' || TRIM(PUT(fci,$fcdir.)) || '/pdf/';
29          CALL SYMPUT('dirloc',TRIM(dirloc));
30          output_file = TRIM(dirloc) || TRIM(LEFT(SYMGET('nomplume')) ||
31              '.pdf';
32          link = TRIM('<A HREF="' || TRIM(LEFT(SYMGET('nomplume')) ||
33              '.pdf ">' || TRIM(filedesc) || "</a>");
34          CALL SYMPUT('pdfout',output_file);
35          OUTPUT;
36      END;
37      DROP dirloc;
38      RUN;
39  %END;

```

We start to build the record for the table of contents (TOC) table by checking to see if the output being generated needs individual identifiers or not. The section above is for individuals. It starts with a RETAIN statement to keep the file descriptor across records of the individual identifier dataset (&DS). We get the user identification in line 13, then begin the file descriptor in line 15. We append the ID variable from the individual identifier in each record to the file descriptor as long as it hasn't read more than (&LIM) records (lines 16/17).

When the last record in the individual identifier file is read, we create the record for the TOC table. First, if we've reached the limit for the number of individual identifiers (&LIM), an ellipsis is used to indicate an unspecified span of ID's, and the last ID is appended (lines 19-22). We get a date and a time for the output from a datetime value. These are displayed on the HTML index to help users identify their output. This datetime value is also used to determine the age of the output. Files over an hour old (in practice, it works out to a maximum of an hour, fifty-nine minutes) are removed by a confab-controlled cleaning program. Each time the macro is invoked, it also will check for expired files.

The field center identifier is gathered from macro space and used to determine the directory (lines 24-27). The PDF output file name and hyperlink are assembled, and the PDF file name (&PDFOUT) is made available to the rest of the application (lines 28-30.) Finally, the record is written to a temporary SAS table.

The section of code on the following page executes if there are no individual identifiers needed in the description.

```

38  %ELSE %DO;
39      DATA desc;
40      LENGTH output_file $ 100 filedesc link $ 300 fci $ 1 userid $ 8;
41      filedesc = TRIM(LEFT(SYMGET('desc')));
42      userid = SYMGET('userid'); fci = SYMGET('fci');
43      datetime = DATETIME();
44      date = DATEPART(datetime);
45      time = TIMEPART(datetime);
46      CALL SYMPUT('fcidir',TRIM(PUT(fci,$fcdir.)));
47      dirloc = "/users/" || TRIM(PUT(fci,$fcdir.)) || '/pdf/';
48      CALL SYMPUT('dirloc',TRIM(dirloc));
49      output_file = TRIM(dirloc) || TRIM(LEFT(SYMGET('nomplume')) || ".pdf";
50      link = TRIM('<A HREF="' || TRIM(LEFT(SYMGET('nomplume')) ||
51          '.pdf ">' || TRIM(filedesc) || "</a>");
52      CALL SYMPUT('pdfout',output_file);
53      DROP dirloc;
54      RUN;
55  %END;

```

In the case of output without individual identifiers (above), the procedure is simplified because we can use the file descriptor (line 39) exactly as obtained from the macro parameter. All we need to do is to get the user, date, time, and the field center (lines 40-42). Next, we get the directory name, create the output file name and link, and then make the output file name available to the application. No OUTPUT; statement is needed because this DATA step only iterates once as opposed to the one with the individual identifiers, which iterates as many times as there are records with identifiers.

The following section of code takes the record we just generated and adds it to the existing TOC table. The time limit for this field center's output file is enforced here between crontab jobs, to further narrow the window during which the output is available.

```

55 DATA SYSDS.PDFS;
56 SET sysds.pdfs desc (KEEP=datetime date time link output_file userid fci);
57 LENGTH cmd $ 200 cmdstrt $ 8 1 3;
58 now = DATETIME();
59 IF (now - datetime) GE 3600 THEN DO;
60     l = LENGTH(output_file) - 4;
61     cmd = "/bin/rm " || SUBSTR(output_file,1,l) || ".ps";
62     CALL SYSTEM(trim(cmd));
63     cmd = "/bin/rm " || output_file;
64     CALL SYSTEM(trim(cmd));
65     DELETE;
66 END;
67 DROP cmd l cmdstrt now;
68 RUN;

```

The KEEP= option on the SET statement in line 52 makes sure that we don't accidentally accumulate unnecessary variables in the TOC table. Lines 54-62 take care of the housekeeping. We figure out if the file is over an hour old based on the difference between the current time and the file timestamp. If so, then the PDF file and any associated PostScript file (if one exists) are removed and the record is deleted from the TOC table. Even if the CALL SYSTEM fails, removing the record from the TOC table makes the PDF file extremely difficult to find on the web server since the actual file name is not readily apparent. In a related note, the TRIM() function was critical in getting the CALL SYSTEM to work on our LINUX machine. The CALL SYSTEM does not work without it, most likely because the command line was too long (padded with spaces) for the shell to process it. Finally, in line 61, we remove the record associated with the deleted file(s) from the TOC table.

The code below represents the easy part: producing the HTML file that the users see. First, the TOC table is checked to see if there is any output available for the given field center. There are multiple ways to do this, including using %SYSFUNC() and the SCL function VARSTAT(), but this method was just as easily coded. If there is no output available for the field center, then a static HTML file is copied to the index.html file (lines 79-84). If we were to use the generated "index.html" file, it would display the results of a PROC PRINT for a table with no records, which is to say, a completely blank HTML page.

```

69 PROC SORT DATA=sysds.pdfs;
70 BY date time;
71 RUN;
72
73 PROC FREQ DATA=sysds.pdfs NOPRINT;
74 TABLES fci / OUT=obschk (KEEP=fci count);
75 WHERE fci EQ "&fci";
76 RUN;
77
78 %LET nout=0;
79 DATA _NULL_;
80 SET obschk;
81 CALL SYMPUT('nout',count);
82 RUN;
83 %IF &nout=0 %THEN %DO;
84     DATA _NULL_;
85     cmd = "cp &dirloc/nofiles.html &dirloc/index.html";
86     CALL SYSTEM(trim(cmd));
87     RUN;
88 %END;

```

If there is output, however, we associate a CSS style sheet with the HTML file in the ODS HTML statement by using the HEADTEXT= option (line 89). The macro variable &DIRLOC creates the HTML file in the directory for the field center of the current user. The WHERE statement in line 94 restricts the display to the records generated by all users from the field center of the current user. The net result is that users from one field center don't even see that output from other field centers even exists.

```

89 %ELSE %DO;
90 TITLE "Print Files For GOLDN Study";
91 FOOTNOTE;
92 ODS LISTING CLOSE;

```

```

93   ODS HTML FILE="&dirloc/index.html"
      HEADTEXT='<link rel="stylesheet" href="gdn.css">';
94   OPTIONS LS=256;
95   PROC PRINT DATA=sysds.pdfs LABEL;
96   ID date time userid;
97   VAR link;
98   WHERE output_file CONTAINS ("%&fcdir");
99   RUN;
100  ODS HTML CLOSE;
101  ODS LISTING;
102  %END;
103  %MEND getdesc;

```

WHAT ABOUT ODS PROCLABEL=?

We could have pursued the strategy of making the PROCLABEL= option function dynamically, assembling file descriptors by using macro variables in the PROCLABEL= option; however, that would not have solved all of our problems. PROCLABEL= only works with ODS-generated output. Since the problem of naming files at run-time would still exist, some variant of this solution would have been necessary.

HOW IS IT USED?

The macro is called immediately before any output is generated. This macro requires the use of a SUBMIT block in the application SCL. In the following example, we are going to create a table of contents entry for a set of clinic and lab labels generated for multiple people.

```

SUBMIT CONTINUE;
%getdesc(desc=Clinic and Lab Labels,ds=label_lst,lim=2);
ENDSUBMIT;

```

Any output generated after this call will have the descriptive heading, “**Clinic and Lab Labels for ID *nnn*, *nnn*... *nnn***”, where “*nnn*” is taken from the variable ID in the dataset LABEL_LST. The value of ID for the first two records of LABEL_LST will be used to fill the specifics before the ellipsis (...), while the value for the last record will end the description. If there are fewer than three records in the dataset, then no ellipsis will appear. The parameters set by this invocation of the macro remain in force until the next invocation of the macro. Therefore, this macro needs to be used before each piece of output is generated. Otherwise, successive output will overwrite (or append, depending on the options used in generating the output) any other output produced after this macro has been called. This is because the physical filename of the output file, as well as the description, remain the same.

This example prints a standardized report. We'll include the ODS statement here.

```

104  SUBMIT CONTINUE;
105    %getdesc(desc=Recruitment Summary)
106    ODS LISTING CLOSE;
107    ODS PDF FILE="&pdfout";
108    PROC REPORT DATA=ind_recsum REPORT=prgm.programs.ind_recsum NOWD;
109    RUN;
110    ODS PDF CLOSE;
111    ODS LISTING;
112  ENDSUBMIT;

```

Any PDF output generated after the PROC REPORT will have the same file name and TOC descriptive text, so the macro will need to be called again before generating any more output. Line 4 is of note; since the application creates non-ODS output, the actual PDF file name has to be passed to the ODS statement because the symbolic file reference is being used for the PostScript file name for non-ODS output. If all output were being produced through ODS, you could use a symbolic file reference by using the value of &PDFOUT in a FILENAME statement or SCL function.

Here's a final example from the application. In this case, the PDF file is not produced by ODS.

```

1  SUBMIT CONTINUE;
2    %getdesc(desc=Family Recruitment Sheets, ds=famtempy, lim=3);
3  ENDSUBMIT;
4  psfn = TRIM(PUTC(fc,'%fcdir.')) || "/pdf/" ||
      TRIM(LEFT(SYMGET('nomplume')) || ".ps";
5  rc = FILENAME('out',psfn);
6  syscmd = "ps2pdf " || TRIM(psfn) || " " || SYMGET('pdfout');
7  SUBMIT CONTINUE;
8    %INCLUDE "/include/recruit_indv.sas";
9  ENDSUBMIT;

```

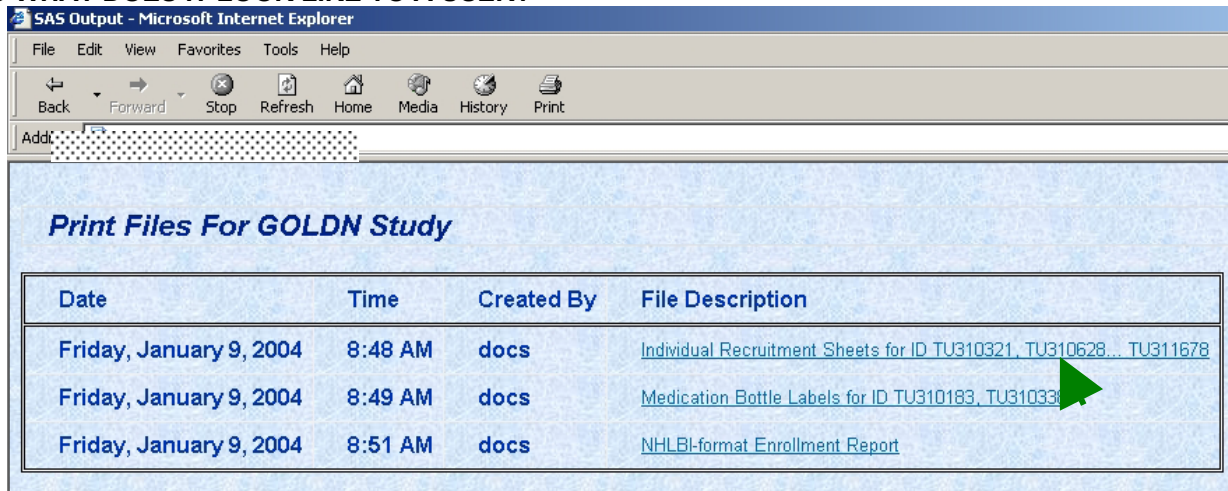
```
10 rc = SYSTEM(syscmd);
```

As with any invocation of this macro, the order in which things are done is extremely important. First, the macro is executed. This is going to use a maximum of three individual identifiers in the descriptor, and the identifiers are taken from the values of the ID variable in the FAMTEMPY table. At this point, FAMTEMPY exists, and has been created from a selection list in the application. It is also used to create the table that the output generation program (in this case, recruit_indv.sas) works with.

Once that has finished processing, the application regains control, and we assemble the PostScript file name, which is the same as the PDF file name with a ".ps" extension rather than a ".pdf" extension (line 4.) We could not do this any sooner, because the unique portion of the file name does not get defined until after the macro has executed. It is available to the rest of the application from the macro variable &NOMPLUME.

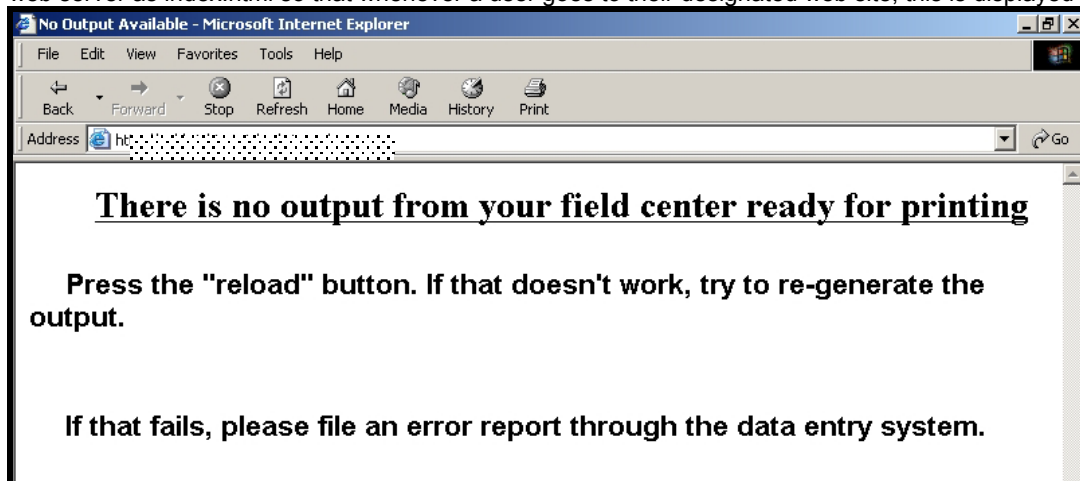
The FILENAME() function is used to define the symbolic file reference for the PostScript file. In line 6, the system command to create the PDF file from the PostScript file is created and stored in an SCL variable. At this point, everything is correctly set up to run the program that generates the output (line 8). This external SAS program uses the FILENAME we created in line 5 ("out") so that we do not have to pass the actual file name (via macro or other means) to it. Once that program has executed and the PostScript file is created, the SYSTEM() function uses the LINUX utility to create a PDF from the PostScript file. The macro has already created the macro variable &PDFOUT, as well as a record in the TOC file for that PDF file.

SO WHAT DOES IT LOOK LIKE TO A USER?



Date	Time	Created By	File Description
Friday, January 9, 2004	8:48 AM	docs	Individual Recruitment Sheets for ID TU310321, TU310628... TU311678
Friday, January 9, 2004	8:49 AM	docs	Medication Bottle Labels for ID TU310183, TU31033
Friday, January 9, 2004	8:51 AM	docs	NHLBI-format Enrollment Report

As you can see, the date, time, application user id, and the file description are displayed on a web page. The PDF output is obtained by clicking on the description text (green arrow), which is hyperlinked to the PDF file. This page is set up on the web server as index.html so that whenever a user goes to their designated web site, this is displayed automatically.



There is no output from your field center ready for printing

Press the "reload" button. If that doesn't work, try to re-generate the output.

If that fails, please file an error report through the data entry system.

This is the message that gets displayed when there is no application output available. It is stored on the web server as a static HTML page, but is never accessed that way. If there is no output, lines 79-84 of the macro code (see p. 4) copies this file to index.html.

SUMMARY

Adequately describing the contents of, and providing easy navigation to, web-served output may be handled with ODS PROCLABEL. With a little bit of creativity, this statement can be dynamically executed to provide more descriptive titles than the standard SAS procedure labels in the default ODS HTML table of contents. However, if the application generating the output uses non-ODS methods, or if multiple users can generate the same type of output using different data, or if the output needs to be routed to different locations on a web server, there is no current tool in ODS that will handle all of these situations.

The macro %GETDESC creates a unique file name for all pieces of output and makes it available to the rest of the application so that each piece of output can be described thoroughly, routed to the appropriate location, and made easily accessible to the person who generated it. PDF files insure the WYSIWYG nature of the output regardless of the hardcopy output device. The macro also has some security features, including a used file cleaner that erases an output file between one and two hours after it has been generated. This macro has made it possible for a non-web Internet-based application to serve the hardcopy needs of remote locations without requiring any more than access to a web browser and local printer somewhere within that location.

REFERENCES:

Morgan D. "Multi-Center Study Data Management With A Distributed Application,"
<<http://www2.sas.com/proceedings/sugi28/158-28.pdf>> (January 15, 2004)

Morgan D. "A PostScript Macro Library," Proceedings of the Seventeenth Annual SAS® Users Group International Conference, 1992, 1049-1054

Morgan D, Province, M. "Simplifying SAS® Security," *Proceedings of the Twenty-Seventh Annual SAS® Users Group International Conference*. April 2002 <<http://www2.sas.com/proceedings/sugi27/p103-27.pdf>> (June 15, 2003)

ACKNOWLEDGEMENTS

This work has been partially funded by NHLBI grant UY1 HL72524-01.

CONTACT INFORMATION:

Further inquiries are welcome to:

Derek Morgan
Division of Biostatistics
Washington University Medical School
Box 8067, 660 S. Euclid Ave.
St. Louis, MO 63110
Phone: (314) 362-3685 FAX: (314) 362-2693
E-mail: derek@wustl.edu

This paper, a Windows version of the code in this paper, and other SAS System examples and papers can be found on the World Wide Web at:

<http://www.biostat.wustl.edu/~derek/sasindex.html>

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

Other brand and product names are trademarks of their respective companies.