

SAS Software Macros- You're Only Limited by Your Imagination, Part II (More Advanced Techniques)

Peter Parker, US Dept. of Commerce, Washington, DC
Peter.Parker@ita.doc.gov

Abstract

SAS software macros provide the flexibility to perform advanced programming techniques such as processing real-time user-entered parameters to determine which blocks of code will run and how often. In my previous paper (Part I), I described how to use these techniques to write more elegant code. In this paper, I will show how to apply these techniques to read multiple external files based on a user query.

Specifically, I will use a real life application and show how it was implemented, using a tightly written program comprised of macros. I needed to determine how often the public accessed the files on my web server. Through iterations of programs that were the evolution of a thought process, I have developed SAS programs that read the web server's log files based on the time periods requested in an interactive SAS macro window. These programs can generate a summarized report of web usage.

This paper is a learning tool for the professional programmer on conceiving and generating a computer program, using SAS Base and macro software. Those who also want to track hits to their web servers will find this information useful.

Introduction

In my previous paper (Part I), I explained how to use SAS macros to create user interfaces (Macro Windows), how to store common code in code libraries, how to prepare top-down structured programs using macro modules and how to run batch processing with specialized SAS icons. In this paper, I take some of these SAS macro techniques and apply them to a real application, tracking web page usage.

As a learning tool, I will first analyze the need for this web site tracking application. Next, I will discuss different approaches to writing it and finally I will explain how this computer program evolved into its final form. Not only will programmers be able to see how SAS macros can be used creatively, but also how a concept becomes a production application.

What is the problem?

We maintain our own web server. We create web applications and perform all necessary system administration on that box. As application developer/analysts, we need to know what people want from our web page. If certain web pages are used more than others, we will devote more resources to maintaining the information on those pages. If pages are little used, we should either drop them, moving those resources into maintaining more popular pages, or we will need to improve those pages to encourage more usage. The number of web page hits will assist us in deciding how to allocate our resources.

Additionally, we need to know if users are accessing restricted areas, particularly those intrusions with bad intentions. Some malevolent users like to hack into web servers, damage files on them and then try to intrude beyond the web server into internal server systems. Besides setting up security measures to stop these intrusions, we need to know if these measures work.

To track web server usage, good or bad, we can skim the daily-generated log files.¹ These files are at least ½ megabyte each. One can use a text editor and check these files, but that would be a tedious process. Some log files have almost 50,000 records. One easily could miss some signs of suspicious activity. As for determining popular web pages, it would be too much information to absorb and to tabulate. For handling, summarizing, and analyzing lots of information, SAS software is the solution.

How to Proceed (The Evolution of the Analysis)

As first, I would check the log files manually. Using a text editor, I could open these files and check for obvious intrusions, such as a "cmd.exe" commands with return codes of 200 (meaning the query was successful). Since this type of checking is tedious, it

¹ We're using Microsoft Windows NT 4.0 operating system, with Internet Information Server (IIS) 4.0. We plan to update to Windows 2000 with Internet Information Server 5.0 during Summer 2002.

was done only occasionally. Moreover, this eyeballing technique was not reliable. With luck, one may find an intrusion attempt or even an intrusion breach. However, a few years ago, when the web server was first set up, security was not a major concern. Web page intrusions, unlike now, were rare.

There always has been a need to record the web page hits to justify maintaining the web page, as well as, to determine how to allocate resources for updating the information in these pages. Initially our counting was performed using Perl software. Whenever a web page was hit, a Perl application would increment a number in a text file. Each web page had its own count in its own text file. Later I developed a Lotus Notes application that would take the data from these text files and put them in a Lotus Notes document. Once a month I would create a new document with the latest counts. I then created an agent that would export these documents to a spreadsheet and create a monthly and cumulative hits display for each type of web page, including line graphs for showing a time-series.

This Perl/Lotus Notes/spreadsheet system had many shortcomings:

1. It was complicated. It had too many steps.
2. It was inflexible. Usually I ran it at the end of a month to create monthly comparisons. If I forgot to run it at the end of the month, I would be missing a month's hits records.
3. It was unreliable. The information is discrete. In each Perl-generated text files, I only have one number. What if that file became damaged? Then I would have to start over again. For one month, Perl was not running properly. My counts became more unreliable.
4. It was incomplete. All I had were total hits. I did not know when the hits were made and by whom.
5. It was difficult to maintain and created an additional layer of complexity. I have to learn Perl, build my skills in it, and administer it on the web server. Ideal web security administration is running as few services as possible. More services means more potential security holes.

Originally, I decided to use Perl because I anticipated creating other Perl applications. These applications never materialized. After the tweaking of some settings on the web server brought down Perl, I decided to re-evaluate how to track hits. Additionally, security concerns have increased dramatically in the past year, especially for government websites. It became apparent that the current methods of tracking hits and intrusions were insufficient. I needed better information.

The solution was obvious. All of the information that I needed was stored in the logs files on the web server. Even the information that I thought was lost when Perl was not running on the web server was still available. I had tried before to use over-the-counter software to analyze the logs and generate reports, but I found them hard to use and inadequate for my needs. Why depend on someone else's clunky front end to the logs files when I could create my own custom log files checks using SAS software?

Types of reports

Web Intrusions- This report is the most critical for security's sake, although once an intrusion is discovered, it may be too late. The user already may have damaged the web server, modified or deleted the data, or may even have used the server entry as a means for further intrusions. However, checking the log files, I can determine whether security has been compromised and then I could decide on necessary actions to fix the damage to the web server and to prevent future successful attacks. Those actions are beyond the scope of this paper.

In an IIS web server, using Windows NT, a new log file is generated every day. The name of the file is "EX<yymmdd>.LOG", where yymmdd is the date. For example, a log file for May 4, 2002 would be called EX020504.LOG. Every entry in that file will be a record of successful and unsuccessful web queries on the web server. A three digit code determines the status of the query; a code value of 200 means that the query was successful. Those successful entries are the ones I want to check. Any other code value was an unsuccessful query, be it legitimate or not. I'm not concerned with them because if they were hacker attacks, they didn't succeed. Besides unsuccessful queries, I also filter out the following:

1. my office, U.S. Dept. of Commerce entries (using the IP address range)
2. local entries (IP address 127.0.0.1)
3. legitimate queries (files with suffixes like .htm, .pdf, .txt, and executables that we've written).

After stripping unsuccessful hits (code > 200) and successful but legitimate hits, I'm left with a small list only of questionable entries. This suspicious hit report will contain entries showing what html command was entered, the user's ip address and the time the data request was made. In example 1 below, observations 1-3 are normal "noise" in the web logs. However observation 4 is a hacker intrusion. The web user was able to run the "cmd.exe" command on the web server.

Example 1. Intrusion detection in Log file of May 19, 2002

WEB LOG CHECK- 020519

Obs	userid	page	time	method	param
1	192.111.222.40	/	18:48:55	GET	200
2	192.111.222.11	/fedregs/	18:49:34	GET	200
3	192.111.222.40	/scripts/ss	18:49:37	GET	200
4	192.111.222.40	/scripts/../../winnt/system32/cmd.exe	18:49:40	GET	200

Rather than wading through thousands of records in the web log and likely missing particular entries, I analyze a few lines of particular interest. In this case, hacker activity was identified.

Hit Activity- This report is crucial for measuring the usefulness of the web page to clients. A measurement of usefulness is how often people hit (i.e., query) those web pages. If a web page has few hits, either the page is not well known or is not useful. Management may decide to allocate more resources to improving those pages and a subsequent increase in web hits may validate that decision. On the other hand, a stagnant count of hits may give the signal to drop those pages. Regardless of how one decides to allocate web page data and programming resources, the number of hits is the main determining criterion.

The program originally read the raw number of hits per web page. However that technique inflated the numbers since a web user may hit on several items within a web page, each one counting as a hit. For instance, if you have a page of different textile imports reports by type of clothing, each report would count as a hit. But if you're concerned only with hits to the textiles imports report, you can record visits instead of hits. I defined a visit as hits to a particular web page from a unique IP address within in certain time period, usually an hour or less. Every web user has an IP address. I furthered modified it based on the hour of the hit. Any hits by a particular IP address between 3:00 a.m. and 3:59 a.m. would be considered a visit. Additionally, I identified the major users of our web pages and I produced a report on the same database on visits by IP address, rather than total visits per web page. Once the data is in an SAS dataset, one can manipulate and produce reports based on any combinations of fields.

Complications-

While SAS software is easily learned and applied by programmers, sometimes complications arise beyond the scope of the normal Base SAS algorithms, which will require clever use of SAS macros. Here are some complications to consider.

1. Everyday a new web server log file is created. In a year, 365 files will have been

created, with the name of the file has the date built in. How do you reference a particular log file without having to edit the SAS program each time?

2. How do you read multiple log files at a time? For instance you may want to see all of the web hits for April 2002. That would require reading in 30 log files. You may want to see hits for a particular week, month, year to date, calendar year or year ending. You could code in each external file in separate filename and set statements but that would be more tedious coding.
3. In addition, suppose you have a range of dates you want to see hits, like February 5, 2002 through June 17, 2002. How do you read multiple log files for an irregular time period?

There are many solutions to reading multiple external files. One solution is the brute force approach. Each file reference in a filename statement would have to be manually entered in the program and then referred to in a set statement of a data step. Not only is this method crude but also it wouldn't make a good SAS conference paper. The preferred and elegant solution to these complications is to read multiple external log files in a do loop, based on parameters passed through macro windows. That will be illustrated in the examples below.

Solution 1- Checking a daily Web log for intrusions.

Every morning I check the previous day's web log for intrusions. I want to make it as simple as possible to use and to read. This program will:

1. query the analyst for a date
2. use that date to specify which web log file to read (if an user specifies May 6, 2002, it will reference the web log file created on that day)
3. filter out unsuccessful or legitimate web queries (html code, specific executables)
4. print a report of suspicious entries.

Here is the basic structure of the program, annotated:

Example 2- Program for checking web log for Intrusion

* **Basic housekeeping;**

```
QUIT;  
RUN;
```

```
OPTIONS MPRINT; *useful for debugging, displays in the log-SAS statements generated by macro execution;
```

```
OPTIONS MLOGIC; *also useful for debugging macros during execution, especially nested macros;
```

```
OPTIONS SYMBOLGEN; *displays results of resolving macro variable references;
```

```
%GLOBAL YEAR MONTH DAY; *defining macro variables as global- can be referenced throughout program;RUN;
```

***Macro Windows to prompt analyst for which log file to read;**

```
*display macro window on screen, so that user can input dates into macro variables YEAR, MONTH, and DAY;
```

```
%WINDOW INITVAL COLOR=BLUE
```

```
#5 @5 "OTEXA LOG FILE CHECK" @50 "&sysday, &sysdate.."
```

```
#10 @10 "LOG FILE DATE-"
```

```
#15 @15 "YEAR? (YY)" @30 YEAR 2
```

```
PROTECT=NO ATTR=HIGHLIGHT COLOR=YELLOW REQUIRED=YES
```

```
#17 @15 "MONTH (MM)" @30 MONTH 2
```

```
PROTECT=NO ATTR=HIGHLIGHT COLOR=YELLOW REQUIRED=YES
```

```
#19 @15 "DAY (DD)" @30 DAY 2
```

```
PROTECT=NO ATTR=HIGHLIGHT COLOR=YELLOW REQUIRED=YES;
```

```
%DISPLAY INITVAL;RUN;
```

***read and filter data;**

```
FILENAME IN1 "X:\ex&YEAR&MONTH&DAY.log";
```

```
*pass date values from macro variables year, month and day. For May 6, 2002, this statement will become-FILENAME IN1 "X:\ex020506.log";
```

```
*Read in log file and strip off unsuccessful hits (param not equal to 200) and legitimate hits (htm and gif files, specific executables;
```

```
DATA ONE;
```

```
INFILE IN1 ;
```

```
length userip $25 page $90;
```

```
input time $ userip $ method $ page $ param ;
```

```
if USERIP= "127.0.0.1" THEN RETURN; *filter out local queries;
```

```
IF INDEX(UPCASE(PAGE),'.GIF') > 0 OR INDEX(UPCASE(PAGE),'.HTM') > 0 OR  
INDEX(UPCASE(PAGE),'.OTEXA.EXE.') > 0
```

```
THEN RETURN; *filter out legitimate queries;
```

```
IF 0 < PARAM < 300 THEN OUTPUT; *include only successful queries;
```

***print out report of suspicious successful queries;**

```
run;
```

```
PROC PRINT;
```

```
TITLE "WEB LOG CHECK- &year&month&day";
```

(See example 1 for how a sample report may appear)

Solution 2- Tracking hits to the web site for a specified period of time.

The next application is counting legitimate hits, to analyze how much and how the web site is used. The same core program as in [Solution 1, Checking for web site intrusions](#) is used with some major modifications. The data still has to be filtered, including only html and executable files with a

successful condition code of 200 and then the results need to be categorized, summed up and printed in a report.

The complication is reading several external log files, after entering a range of dates when prompted by a macro window. This solution involves using do loops in executing macros, as described below:

Example 3- Program for Tracking Web Hits

```
.*
.*
.*
*Macro Windows to determine which log file to read;>
*note that that 2 dates are requested, beginning date and ending date;
%GLOBAL YEARMODAY BEGINDATE ENDDATE BDATE$AS EDATES$AS DATES$AS;
*defining macro variables as global- can be referenced throughout program;
*macro variables Begindate and Enddate contain the user-inputted beginning and ending dates;
RUN;
%WINDOW INITVAL COLOR=BLUE
#5 @5 "OTEXA Web Counters" @50 "&sysday, &sysdate.."
#10 @10 "LOG FILE DATE-"
#15 @15 "Beginning date? (mmddyyyy)" @45 Begindate 8
PROTECT=NO ATTR=HIGHLIGHT COLOR=YELLOW REQUIRED=YES
#17 @15 "Ending date? (mmddyyyy)" @45 Enddate 8
PROTECT=NO ATTR=HIGHLIGHT COLOR=YELLOW REQUIRED=YES;
```

***Macro INPUTDATA is the module to read and filter the data. It will be called by the macro LOOPDATA in a do loop;**

```
%MACRO INPUTDATA;
FILENAME INDATA "x:\EX0%left(&yearmoday).log";
*dummy data one for errors to prevent double counts;
DATA ONE;
LENGTH PAGECAPS $90 ;
PAGECAPS=" ";
COUNTS= 0;
RUN;
*.;
DATA ONE (KEEP= PAGECAPS COUNTS);
INFILE INDATA;
length userip $25 page $90;
input time $ userip $ method $ page $ param ;
IF 0 < PARAM < 300 ;
COUNTS= 1;
PAGECAPS= UPCASE(PAGE);
*filter out gif and jpg files;
IF INDEX(UPCASE(PAGE),'GIF') > 0 OR INDEX(UPCASE(PAGE),'JPG') > 0 THEN RETURN;
*filter out local hits;
IF USERIP = "127.0.0.1" THEN RETURN;
*include hits for files types I want to count;
IF INDEX(UPCASE(PAGE),'STM') > 0 OR INDEX(UPCASE(PAGE),'HTM') > 0 OR
INDEX(UPCASE(PAGE),'EXE') > 0 OR INDEX(UPCASE(PAGE),'STM') > 0 OR
INDEX(UPCASE(PAGE),'HTM') > 0
THEN OUTPUT;
run;
%MEND INPUTDATA;
```

***Macro LOOPDATA will read in macro INPUTDATA until all web logs have been read, based on beginning and ending dates entered in macro window;**

```
%MACRO LOOPDATA;
Data _Null;
Length BMonth EMonth $2;
Length BDay EDay $2;
Length BYear EYear $4;
BMonth=%Substr(&BeginDate,1,2);
EMonth=%Substr(&EndDate,1,2);
BDay=%Substr(&BeginDate,3,2);
EDay=%Substr(&EndDate,3,2);
BYear=%Substr(&BeginDate,5,4);
EYear=%Substr(&EndDate,5,4);
call symput('BDate$AS',mdy(BMonth,BDay,BYear));
call symput('EDate$AS',mdy(EMonth,EDay,EYear));
run;
data _null_;
call symput('yearmoday',((year(&Bdate$AS)-2000)*10000)+(month(&Bdate$AS)*100)+day(&Bdate$AS));
run;
%INPUTDATA;
RUN;
DATA TWO;
SET ONE;
```

```

RUN;
%LET DATESAS=&BDATESAS;
%DO %WHILE(&DATESAS < &EDATESAS);
  %LET DATESAS=&DATESAS+1;
  Data _Null;
  call symput('yearmoday',((year(&DateSAS)-2000)*10000)+(month(&DateSAS)*100)+day(&DateSAS));
  run;
  %INPUTDATA;
  DATA TWO;
  SET TWO ONE;
  RUN;
%END;
%MEND LOOPDATA;
RUN;

```

***Macro PROCESS will summarize data and categorize it;**

```

%MACRO PROCESS;
PROC SORT DATA=TWO; BY PAGECAPS;
PROC MEANS SUM NOPRINT;
BY PAGECAPS;
VAR COUNTS;
OUTPUT OUT=COUNTER SUM=COUNTS;
RUN;
DATA CTRSUM;
SET COUNTER;
LENGTH PGLABEL $ 30;
SELECT;
WHEN (INDEX(PAGECAPS,'BILAT') > 0) PGLABEL="BILATERAL AGREEMENTS";
WHEN (INDEX(PAGECAPS,'CORRELAT') > 0) PGLABEL="CORRELATION";
WHEN (INDEX(PAGECAPS,'EXPORTADVANTAGE') > 0) PGLABEL="EXPORT ADVANTAGE";
.
.
END;
RUN;
PROC SORT DATA=CTRSUM; BY PGLABEL;
PROC MEANS SUM NOPRINT;
BY PGLABEL;
VAR COUNTS;
OUTPUT OUT=CTRSUM1 SUM=COUNTS;
RUN;
PROC SORT; BY DESCENDING COUNTS PGLABEL;
RUN;
%LET BEGINDAT1=%SUBSTR(&BeginDate,1,2)/%Substr(&BeginDate,3,2)/%substr(&BeginDate,5,4);
%LET EndDAT1=%SUBSTR(&EndDate,1,2)/%Substr(&EndDate,3,2)/%substr(&EndDate,5,4);
RUN;
PROC PRINT;
VAR PGLABEL COUNTS;
FORMAT COUNTS COMMA10.;
SUM COUNTS;
TITLE "OTEXA WEB HITS &BEGINDAT1 - &ENDDAT1";
RUN;
%MEND PROCESS;

```

***Macro DRIVER (like in top down structured COBOL programming) where each module of the program is executed;**

```

%MACRO DRIVER;
* prompt user for beginning and ending dates in macro window;
%DISPLAY INITVAL;
RUN;
* read in data from log files, depending on beginning and ending dates;
%LOOPDATA;
RUN;
*process this data for a report;
%PROCESS;
RUN;
%MEND DRIVER;
%DRIVER;
RUN;

```

Solution 3- Tracking visits (hits within an hour) for a specified period of time.

Suppose a web user hits several pages within a section, for instance different phone numbers in the office personnel page. You may not want to count each of these hits individually. Rather, you may prefer to count "visits" for each type of section. My precise definition is hits by an IP address for a defined category (personnel information, monthly commodity imports reports, etc.) for a defined period of time. For simplicity, I created a variable that strips off the hour of the query. All queries for IP 133.133.133.133 between 9:00 a.m. to 9:59 in the personnel files will be counted as one visit. If that same IP also queries the files at 10:02 a.m., it will count as another visit.

Solution 4- Other quick programs

Once the basic program is written to read multiple external files, one can make many variations on it. I have made other programs from this core that do the following:

1. Report by IP address to determine who visits the web page the most.
2. Report web pages visited by a particular IP address- If I've discovered that a person is trying to intrude on my web server (e.g., a "cmd.exe" appears in the log file) I want to track all of his activity for that day, especially successful queries.
3. Create a time series chart/graph on particular web pages to determine what future web usage may be.

Conclusion

There's two ways to write code, either with brute force or with elegance. I prefer the latter. Both methods will produce usable reports, but the former will require a lot more typing and debugging. Just reading in a year's worth of log files would require typing in 365 lines of file references, as well as several lines of tedious code in the set statement.

Elegant code, if a programmer has enough time, is always preferred to brute force code. Referencing a years worth of log files requires only entering the correct beginning and ending dates. The program will literally run itself.

Elegant code writing is good code style. It is brief, to the point and easy to read. As illustrated in this paper, using macros and macros windows, one can

write good code with style. One is only limited by one's imagination.

References

Art Carpenter, *Carpenter's Complete Guide to the SAS Macro Language*, Cary, NC: SAS Institute Inc., 1998 242 pp.

Parker, Peter (2000) "SAS® Software Macros: You're Only Limited By Your Imagination" in SESUG Conference Proceedings and NESUG Conference Proceedings. Cary, NC: SAS Institute

SAS Institute Inc., *SAS Guide to Macro Processing*, Version 6, Second Edition, Cary, NC: SAS Institute Inc., 1990, 319 pp.

SAS Institute Inc., *SAS Macro Language: Reference*, First Edition, Cary, NC: SAS Institute Inc., 1997. 304 pp.

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 registered trademarks or trademarks of their respective companies.