INTRODUCTION

Imagine that you have just gotten a request from your boss to generate a trend of daily CPU busy averages by shift for a particular computer system. Also, imagine that the only source of this data, historically, is its depiction via low resolution bar charts on listings that have been generated and archived daily for some time (the original data has long since been discarded). Your challenge is to find a way to recreate the data from its bar chart depiction on the archived listings. The SAS System of software offers a very rich programming language that can be used to solve this complex programming problem. This paper demonstrates the power of the SAS programming environment by solving this problem using the data step, arrays, a variety of functions and PROC TRANSPOSE. In addition, an example of using boolean logic within a mathematical expression is shown.

Sometimes the demands of a data processing task require data to be obtained from very non-standard sources. In the SAS System, it is always most favorable to get data from SAS data files or via SAS data engines. If this is not possible then it is pretty easy to write a SAS data step program to read the data from a standard non-SAS file: sequential, partitioned, xam, and a variety of other file/record organizations. The only real difficulty in these cases is that the programmer must know the layout of the fields on the record so to be able to tell the SAS System this information on the INPUT statement. The data step program can then read the file and create a SAS data set of the desired data. But sometimes the desired data is in a very non-standard source such as consoles, listings and logs. These can be difficult because the data in question may be non-linear and non-uniform in position and surrounded by a multitude of trivial characters. At the extreme case, the data may not be represented by a number or word to extract but is, instead, displayed as a picture such as a bar chart.

The SAS System lends itself readily to the extraction/retrieval of even the most stubborn of non-standard data. The SAS data step programming language supplies the computer user with all the necessary constructs (statements and functions) for reading, manipulating and analyzing a source to get the desired data. In this example, the source of the data is a vehicle bar chart of CPU utilization by time of day. One chart represents one day's worth of data. Each bar is subgrouped by workload where all occurrences of an associated character for a specific workload would indicate that workload's contribution to the overall utilization level of the CPU during that time of day interval. A SAS program was written to read the listing, identify and output all data points on the chart, transpose the data and determine the values to be assigned to all data points and derived variables.

THE PROCESS

Before writing a SAS program to read data from a non-standard data source, the contents of a representative sample of the input data source must be fully reviewed and understood. The program will be scanning the source on a line by line basis and searching for clues that indicate the start and end of those sections of the source that contain the data. The sum total of these clues must be unique to the desired section of the source. If the review indicates that there can be assumptions made about the fixed nature of that section of the data source (the more fixed the better) then the program should test these assumptions each time it reads from the source (non-standard data sources can change in non-standard ways). Once the clues (flags) to the section of the source that contain the data have been established, a manual effort of extracting the data can be employed. This gives a clearer idea to the activities that the program will be required to perform to accomplish the same objective (though much, much faster).

Appendix B shows the section of the data source that contains the desired data for this application. It is one of many bar charts in the same listing. Column 1 of the listing contains carriage control characters. There is one chart per page. So a "1" in column one indicates the start of a new chart. The first non-blank line after the page break line contains some of the desired data (NODE, VAXTYPE). The first 18 non-blank characters are the same on every chart. This line is always followed by another line that contains the date of the data. At the top of each bar chart is a line that indicates what the bar chart is charting. In this example we are only interested in the bar charts with the 'CPU utilization' title. This line is always followed by a line that contains the Y-axis title ('%used' in this case) and a series of hyphens that start in the same column at the bar chart title. The Y-axis line is indicated with exclamation points and the values always go from 5 to 100 by 5. The X-axis contains hyphens and numbers. They are always in the same order and length (each character position represents 20 minutes) and the bars only use 7 different characters (6 indicated in the top legend and 1 indicated in the bottom legend).

If this data were to be extracted manually, the following actions would have to take place:

1. The start time of the time of day intervals would go from 00:00 to 23:40 incrementing by 20 minutes from bar to bar.
2. The CPU busy value for a given bar would be equal to the y-axis value at the same height of the highest character on that bar.
3. A workload's percentage contribution to that period's CPU utilization level would be equal to the number of occurrences of it's associated character divided by the number of non-blank characters in that bar (times 100).

Using the SAS System, a program was written to obtain this information from the listing, saving many hours of manual labor (there were several months of daily charts) and providing management with the information it needed in a timely manner.

THE PROGRAM

Appendix B is a complete listing of the program used to extract the required data from the bar chart. It contains 4 basic sections:
1. Read the bar chart listing to create a data set that contains one observation for each Y-axis data point line.
2. Transpose that data set so that there is one observation for the Y-axis values and one observation for each time of day interval.
3. A data step that reads the transposed data set, retains the Y-axis values and uses arrays with do-groups to determine the values for CPUBUSY and each workload's percent of the total processor utilization.
4. Append the new data to the permanent library data set.

The first three program sections will be discussed in more detail.

READ THE BAR CHART

The program begins with a FILENAME statement that defines the macro variable to identify the LISTING file to be read. The first data step defines and retains (and drops) five flags that are initialized to value of FLAG 1--fLAG2 to '1'). The following three tests ensure zero. Two temporary variables are read from each line of the input SUBSTR), the data step tests for the occurrence of particular character string handling functions (INDEX, UPCASE and SUBSTR), the data step tests for the occurrence of particular character string values in TEST2. The first two tests allow the program to obtain values for VAXTYPE and DATE (and sets the value of FLAG1--FLAG2 to '1'). The following three tests ensure that the page of the listing is the one of interest (and sets the value of FLAG3--FLAG5 to '1').

After these tests are complete, no further processing will occur for the current input line if any flag has a value of '0'. If the input line has a non-blank value for TEST1 then all FLAG variables will be reset to zero. With subsequent processing of the input line in the data step, two tests of the line are made to ensure that the Y and X axis are as expected. If either of these tests fail, a message is written to the SASLOG and execution of the program is aborted. If the X-axis is detected and is in its expected form then execution of the data step is stopped (no more data to read and output). The rest of the data step obtains the VALUE of the Y-axis and the character value in each chart data point cell from the current line. Since each column of the chart represents 20 minutes of activity, there are 72 data point cells in each row of the chart (1440 minutes in a day divided by 20 equals 72 intervals). These operations are accomplished using the INPUT and SUBSTR functions, an ARRAY statement and DO-loop processing.

TRANSPOSE THE DATA

Two SAS data sets are created using PROC TRANSPOSE. The first transposition creates a data set whose transposed variables are the value of each of the Y-axis chart values (the VALUE variable). The second creates a data set whose variables represent the value of the data point cell at that particular "level" of the Y-axis (there are twenty value levels on the Y-axis in all, 5--100 by 5). Since there are 72 time intervals on the X-axis, there will be 72 observations in the NEWDATA data set. The value of the _NAME_ variable in the NEWDATA data set will be the name of the DP1-DP72 variable from which the values of the COL1-COL20 variables are derived.

CREATE THE FINAL NEW DATA

The two datasets created from the two invocations of PROC TRANSPOSE are input to a data step that will output a data set with one observation for each time of day interval and will contain the following variables: DATE, NODE, VAXTYPE, TIME, CPUBUSY, NET, INT, BAT, OVH, ITR and OTH. TIME is a SAS time variable that is equal to the time of day interval. CPUBUSY is the average CPU busy for that interval. And NET—OTH are the percents that each of these subtasks (workloads) contributed to the overall system usage. The sum of these percentages is equal to 100. ARRAYS and DO--groups are the primary tools for transforming the transposed data into the final form for output.

When an observation is read from the VALUE data set, the Y-axis values (V1--V20) are assigned to corresponding ARRAY of variables so that they can be retained across multiple iterations of the data step. Then the data step returns for another observation. All subsequent data step processing is only for observations that are read from the input NEWDATA dataset.

The time of day is determined using the _NAME_ variable. The original data point variable names all began with "DP" and ended with a number that indicated the related time of day period (1 is the first 20 minute interval, 00:00:01--00:20:00, 2 is the second 20 minute interval, 00:20:01--00:40:00, and so on). So, to determine the time of day for one interval, it is first necessary to obtain a value for the number of the interval from the _NAME_ variable value. This is done using the INPUT, SUBSTR and LEFT functions. A problem with this operation is that the length of the _NAME_ character string may be 3 or it may be 4. If the SUBSTR function tries to read beyond the end of the character string, an 'illegal argument' error will occur.

One resolution to this would be to use IF--THEN statements to examine the length of the string and to SUBSTR accordingly. In this example, a boolean expression is included as part of the 3rd argument of the SUBSTR function. If the length of the string is 3 then the expression returns a one which is subtracted from 2 to get the correct value for the SUBSTR function length argument. But if the length is not 3 then a zero is returned from the expression and nothing is subtracted from 2. The resulting TIME value is adjusted to represent the beginning of the interval in SAS time of day form (number of seconds since midnight).

The values for CPUBUSY and NET—OTH are initially obtained via a SELECT group within an iterative DO--loop. In order to test for data point values from the bottom of the Y-axis to the top, it is necessary to read the ARRAY elements in reverse order. This is due to the fact that the higher values for the Y-axis occur in the original data set first (100 to 5 by 5). PROC TRANSPOSE encountered these values in that order. So the values of the ARRAY elements are in top-of-the-chart to bottom-of-the-chart order. The value for the COL array variable is examined. If it is 'N' then there is no data for that interval. CPUBUSY is set to missing and the DO--loop processing is stopped for that observation. If it is blank, then CPUBUSY is equal to the Y-axis value (the VAL array) associated with the COL array element referenced by the prior iteration of the DO--loop. It is DAY/BOX/*, then the corresponding workload variable is incremented by 1. After the DO--loop processing is complete, the workload variable values are summed, the percent of total CPUBUSY is calculated for each of the workloads and an observation of output to the NEWDATA dataset.

CONCLUSION

Though nobody wants to ever have to write a program to obtain data from a non-standard source such as a low resolution bar chart, the demands of a data processing job sometimes requires this to be
performed. The SAS system is very capable of dealing with such a situation. This example (taken from a real life situation) shows how this was accomplished via a SAS program so the data needs of a data processing center were met. Once this old information was transformed into data, the CPE analyst was able to use that data with SAS to provide management with new and better information.

SAS is a registered trademark of SAS Institute Inc., Cary, NC, USA.

APPENDIX — A

FILENAME INDATA "ASYSPARM LISTING A";
" READ BAR CHART LISTING
" TO GET DATA POINTS.
" DATA NEWDATA; INFILE IN DATA;
LENGTH NODE $ 5 VAXTYPE $ 4;
RETAIN DATE. ; FORMAT DATE DATE7.;
RETAIN FLAG1-FLAGS '0' NODE VAXTYPE ' , ;
DROP FLAG1-FLAGS TEST1 TEST2 A ;
INPUT@l TEST1 $1.@2TEST2$CHAR82.;
CHECK & READ FOR PROCESSOR TYPE /
IF INDEX(UPCASE(TEST2),
'VPA V2.0 HISTOGRAM')>0
THEN DO;
FLAG1='1';
NODE=SUBSTR(TEST2,37,S);
VAXTYPE=SUBSTR(TEST2,48,4);
RETURN;
END;
CHECK & READ FOR REPORT DATE ,
IF INDEX(UPCASE(TEST2),
'00:00 TO' )>0
THEN DO;
FLAG2='1';
DATE=INPUT(SUBSTR(TEST2,29,7),DATE7.);
RETURN;
END;
CHECK & FLAG FOR CPU USE RPT ,
IF INDEX(UPCASE(TEST2),
'CPU UTILIZATION ' »0
THEN FLAG3='1';
CHECK & FLAG FOR TITLE BOX ,
IF INDEX(UPCASE(TEST2),
'-----')>0
THEN FLAG4='1';
CHECK & FLAG FOR MEASUREMENT ,
IF INDEX(UPCASE(TEST2),
' %USED ' )>0
THEN DO;
FLAGS='1';
RETURN;
END;
RETURN, IF ANY FLAG IS EQUAL 0 ,
IF FLAG1='0' OR FLAG2='0' OR
FLAG3='0' OR FLAG4='0' OR
FLAGS='0'
THEN RETURN;
RESET FLAGS & RETURN IF ANY ,
ADDITIONAL CC DETECTED ,
IF TEST1='w' THEN DO;
FLAG1='0'; FLAG2='0';
END;
TEST FOR Y-AXIS POSITION ,
IF SUBSTR(TEST2,8,3)=' 1' THEN DO;
PUT ' NON-STANDARD CPU'
' UTILIZATION CHART Y-AXIS ?';
LIST; ABORT RETURN 999;
END;
TEST FOR X-AXIS COMPOSITION ,
IF INDEX(TEST2,':0')=0 THEN DO;
IF SUBSTR(TEST2,11,10)="
'01-02-03'
THEN DO;
PUT ' NON-STANDARD CPU'
' USAGE CHART X-AXIS ?';
LIST; ABORT RETURN 999;
END; STOP;
END;
WITH ALL CONDITIONS MET, READ ,
EACH DATA POINT CELL FOR THE ,
CURRENT PAGE LINE AND OUTPUT ,
VALUE=INPUT(SUBSTR(TEST2,5,5),3.);
ARRAY DP [*] $ DP1-DP72 ;
DO A=1 TO 72;
DP[A]=SUBSTR(TEST2,A+10,1);
END;
OUTPUT NEWDATA;
RUN;
WITH ALL CONDITIONS MET,
SORT AND TRANSPOSE DATA SET TO GET ;
VALUE AND DATA POINT ARRAYS.
PROC SORT DATA=NEWDATA;
BY DATE NODE VAXTYPE;
RUN;
OBTAIN Y-AXIS VALUE ARRAY ,
PROC TRANPOSE DATA=NEWDATA
OUT=VALUE PREFIX=V;
BY DATE NODE VAXTYPE, VAR VALUE;
RUN;
OBTAIN X-AXIS VALUE ARRAYS ,
PROC TRANPOSE DATA=NEWDATA OUT=NEWDATA;
BY DATE NODE VAXTYPE, VAR DP1-DP72 ,RUN;
SET-BY DATASETS. USE ARRAYS TO ;
ASSIGN VALUES TO DATA POINTS AND ;
CALC PERCENTS BY BAR (DATA POINT) ;
CHARACTER DEFINED GROUPS.
DATA NEWDATA; SET VALUE(IN=V) NEWDATA;
BY DATE NODE VAXTYPE;
LENGTH DATE TIME CPUBUSY NET INT
BAT OTH ITR OTH 4;
FORMAT TIME TIME5. CPUBUSY 3.
NET—OTH 5.1;
KEEP DATE NODE VAXTYPE TIME
CPUBUSY—OTH
ARRAY COL [20] COL1—COL20 ;
ARRAY VAL [20] VAL1—VAL20 ;
ARRAY VRT [20] VI—V20 ;
RETAIN VAL1—VAL20 0 ;
IF OBS FROM VALUE DATASET ,
THEN ASSIGN VALUES TO ;
RETIRED VALUE ARRAY

IF V THEN DO:
DO I=1 TO 20;
VAL(I)=VRT(I);
END; RETURN;
END;

DETERMINE TIME-OF-DAY FROM
TRANSPOSITION_NAME_VAR
TIME=INPUT(SUBSTR(LEFT_NAME_,
3,4),LENGTH_NAME_));
TIME=TIME/2400-(20/2400);
INITIALIZE VARS & FLAG TO ZERO
CPUBUSY,.o; NET..Q;
INT~;
BAT=<>; OTH=O; ITR=<>; OTH=O; FLAG.'O';
read backWards
across array.
Count number of
each subtype & calc cpu busy.
DO I=20 TO 1 BY -1 UNTIL(FLAG='I');
SELECT ITRIM(COl{I})}:
WHEN ('0') DO:
NET_NET. I: END:
WHEN ('I') DO:
INT .INT.I: END:
WHEN ('B') DO:
BAT.BAT.,: END:
WHEN ('O') DO:
OVH.oVH.,: END:
WHEN ('X') DO:
ITR-lTR.I: END:
WHEN ('N') DO:

OTHERWISE DO:
PUT '1 NON-STANDARD DATA'
POINT ENCOUNTERED 1';
PUT ', ABORT RETURN 998;
END;
END;

IF I==1 AND
COII}'.
THEN CPUBUSY=VAL(I):
END:

TOTAL-SUM(NET,INT,BAT,OVH,ITR,OTH):
ARRAY PCT {S} NET-OTH :

TOTAL*100;
END;
OUTPUT NEWDATA:
RUN:

APPEND NEWDATA TO PERMANENT DATA SET:
PROC APPEND BASE..cAPAPLAN.vAXCDLY
NEW.NEWDATA:
RUN;

VPA V2.0 HISTOGRAM EAGLE (VAX 8820)
30AUG90 00:00 TO 31AUG90 00:00

1302