Using SAS to Automate Statistical Reports

Dr. Madhukar Joshi
Texas Instruments, Sherman, TX 75090

Abstract

Sherman, Texas Semiconductor Facility (SFab) of Texas Instruments is the epitome of "high technology, high volume, and complex" manufacturing operations: over 800 devices are made using numerous technologies, equipment types and processing steps. Nearly 25% of Texas Instruments Inc volume is processed in this facility. Thus, using manual methods is highly impractical — automation of statistical reports and charts is a must. On the other hand, it is not necessary to employ complex or highly sophisticated tools (translation: Expensive) to implement automation. This paper will illustrate how the tools already included in Base SAS and SAS / Graphs can be used to facilitate the necessary automation. Note: Skillful use of these tools is at the core of a recently awarded Patent Award. Specifically, this paper will illustrate how Data _Null_, Proc SQL and SAS Macro statements can achieve the automation.

1. Introduction

Sherman, Texas Semiconductor Facility (SFab) of Texas Instruments is the epitome of "high technology, high volume, and complex" manufacturing operations: over 800 devices are made using numerous technologies, equipment types and processing steps. Nearly 25% of Texas Instruments Inc volume is processed in this facility. Thus, using manual methods is highly impractical — automation of statistical reports and charts is a must. On the other hand, it is not necessary to employ complex or highly sophisticated tools (translation: expensive) to implement automation. This paper will illustrate how the tools already included in Base SAS (Data _Null_, Proc SQL, SAS Macro statements) can be used to facilitate the necessary automation. The author has used these tools to:

- Create spreadsheets (.CSV) from SAS datasets
- Ask SAS to write SAS code that would then generate spreadsheets
- Compute and display month-by-month process capability (Cpk) spreadsheets and trend charts
- Compute and display lot-by-lot trends of lot means and extreme values for each of the many electrical tests performed by device groups
- Develop wafer probe planning reports that distinguish high-yielding wafers from wafers that must be fully probed so as to predict probe cycle times
- Generate HTML code as a "Table" of devices (10 per line) so that double-clicking a device name identifies all electrical tests done on that device and the associated lot-by-lot trend graphs

The corresponding statistical reports and trend graphs are automatically generated early every morning and placed on the company-internal web page so that process-and-equipment engineers can focus on resolving the issues / improvement opportunities highlighted by these. One needs only need Base SAS and SAS FSEDIT modules to make the automation work.

1.1 SAS Data Step

At a minimum, a SAS Data step has the following components:

- A “Data Statement” that describes the name for the data set that will be generated.
- One or more statements to identify the source of data (another dataset, an ASCII file, or a formula for generating the data).
- A “Run Statement” that tells SAS to generate the data set identified by the collection of statements from the “Data Statement” through the “Run Statement”.

A Data statement such as: “Data John;” tells SAS to generate a data set with the name “John”. If one to use the mechanism of a Data step without in fact generating a data set, then one should use the data set name “_Null_”.

1.2 SAS SQL

To invoke SQL in the SAS system, use the following statement.

“Proc SQL;”

An SQL remains active till it encounters a “Quit;”, “Data”, “Proc” or a “Macro” statement. At a minimum, an SQL query can consist of the following statement.

Select <variable list>

From <source dataset>;

This results in a spreadsheet with one column for each variable in the list and one row for each observation in the source dataset. Optionally one can use an SQL to create an output dataset. The resulting dataset or report can be modified using SQL modifiers such as:

Where, Group By, Having, Order By, Left Join, Full Join, or Coalesce.

1.3 Macro Facility

One defines a SAS Macro using the statement

%Macro Name<(Parameter List)>;

The definition ends when SAS encounters a

%Mend;

A macro definition can consist of

Data statement(s), SQL queries as well as other SAS procedures.

Once defined, the macro is invoked using the statement

%Name<(Parameter Values)>;

1.4 Plan of Action

The rest of this paper will illustrate the uses of Data _Null_, SQL and Macro facility for automating statistical reports in specific situations.

2 Exporting Data to a .CSV File

There are three subsections: Specific Month and Year; CSV File for a Generic Month and

2.1 Specific Month and Year

Consider the following situation. The factory data in a specific month and year has some **out-of-spec** observations and one wishes to provide this information to the decision-makers who do not have access to SAS on their PC. To help them track such data, it is necessary to provide some identifying information such as: Lot number, Device name, Technology, Log Point, Equipment used, Operator, Date and Time associated with the out-of-spec data. This can be done using the SAS statements shown in Figure 1.

Figure 1: Create a .CSV File

```
Data _Null_;
  File "/home/account/library/OutFile_03_2004.CSV" lrecl=384;
  Set library.OutFile_03_2004;
  If _N_ = 1
    Then Put "Lot, Device, Tech, LogPoint, Equip, Operator, Date, Time,
    LSL, TGT, USL, Value";
  Put Lot ',' Device ',' Tech ',' LP ',' EQP ',' OPER ',' Date ','
    Time ',' LSL ',' TGT ',' USL ',' RDG;
Run;
```

2.1.1 Notes on Figure 1

- The "Data _Null_;" statement invokes this step and the "Run;" statement ends it.
- The "File" statement is used to generate a file in the user's operating system. The identifier in double quotes is the name of that file in the UNIX environment. The ".CSV" defines this file to be a Comma Separated Values file.
- The "Set" statement identifies the source data set whose contents are to be exported to the file identified in the File statement.
- SAS supplies _N_ as an observation counter.
- The "Put" statement is used to define the contents of the output file. Just prior to listing actual data values, one needs to create the column headers. This is accomplished by the "Put" statement corresponding to "_N_ = 1" condition.
- The next "Put" statement is a loop that starts with the first observation in the data set and ends with the last observation. This statement is used to output the values of variables, Lot, Device, etc.
- The field names (Lot, Device, etc.) are separated by ',' to ensure that the consecutive fields in the output file are separated by a comma (,).
- When SAS processes the above Data _Null_ step, it substitutes the values of Lot, Device, etc. with their values in the data set identified by the Set statement – one observation at a time.
- If the incoming data set has N observations, then the .CSV file will have N +1 lines.
- The decision-maker can open this file with Excel.

2.2 CSV File for a Generic Month

The set of statements in Figure 1 will export the Out-Of-Spec data for the month 03_2004. Figure 2 shows how one can obtain such reports for a generic month.

Figure 2: A Macro Definition

```
%Macro MakeOutFile(Mo_Yr);
   Data _Null_;
   File "/home/account/library/OutFile_&Mo_Yr..CSV" lrecl=384;
   Set LIBRARY.OutFile_&Mo_Yr;
   If _N_ = 1
      Then Put "Lot, Device, Tech, LogPoint, Equip, Operator, Date,
           Time, LSL, TGT, USL, Value";
   Put Lot ',' Device ',' Tech ',' LP ',' EQP ',' OPER ',' Date ','
       Time ',' LSL ',' TGT ',' USL ',' RDG;
   Run;
%Mend;
```

2.2.1 Notes on Figure 2

The statement,

%Macro MakeOutFile(Mo_Yr);

starts the definition of a SAS macro needed to identify the observations to be placed in the OutFile in any generic month, Mo_Yr. The designation, &Mo_Yr, indicates that this is a macro variable whose values would be resolved at run time.

The next 7 statements are copied from their counterparts in Figure 1 – except that the specific month, 03_2004, is replaced by a generic month, &Mo_Yr.

The extra dot '.' after &Mo_Yr is needed because &Mo_Yr is a macro variable.

The macro definition ends when SAS encounters the "%Mend;" statement.

Once defined, it is possible to generate a .CSV file for any month; say 03_2004, by invoking the statement

%MakeOutFile(03_2004);

2.3 Automatic Recognition of the Month

To generate the OutFile reports month by month, one must enable SAS to identify month and year using the system clock. Figure 3 shows the SAS code needed to accomplish this task.

Figure 3: Obtaining Day, Month and Year

```
%Macro Set_DD_MM_YY(Thisday=.);
  %Global Mo_yr;
  Data _Null_;
   Format Thisday ddmmyy10.  Month $2.  Year $4.;
   If &Thisday=.
     Then Thisday = Today() ;
     Else Thisday=&Thisday+0;
    NMonth = Month(Thisday);
    If NMonth < 10
     Then Month = '0'||Left(NMonth);
     Else Month = NMonth;
    Year = Year(Thisday);
    Mo_Yr = Trim(Month)||'_'||Trim(Year);
    Call Symput('Mo_Yr', Trim(Mo_Yr)):
     Run;
%Mend;
%Set_DD_MM_YY;
%MakeOutFile(&Mo_Yr);
```

2.3.1 Notes on Figure 3

- The %Macro statement is used to start defining a macro, in this case, Set_DD_MM_YY. Thisday is a parameter that is either a specific day or it is a dot '.' – a SAS convention to identify missing data.
- The macro variable, Thisday, has a default value of '.' so that one need not specify a value for Thisday when invoking %Set_DD_MM_YY – unless, of course one does wish to invoke this macro on a specific date – for instance, when testing the source code.
- It might be necessary to use the macro variable &Mo_Yr in more than one macro. The %Global statement allows for this.
- The "Data _Null_;" statement starts a data step.
- The Format statement tells SAS that one wishes to use "Thisday" in the form date/month/year where date, month and year are separated by a slash '/' and the calendar year is reported as a four-digit number. It also tells SAS that Month is a 2-character literal and that Year is a 4-character literal.
- The variable, "Thisday" is assigned the value of Today(),a SAS function used to obtain the current date.
- Month is a SAS function to determine the Month associated with Thisday and the Year function obtains the Year corresponding to Thisday.
- Mo_Yr is obtained by concatenating Month to Year. The Trim function eliminates blank spaces, if any. The Call Symput stores the trimmed value of Mo_Yr in a macro variable &Mo_Yr.

- The “Run;” statement completes the data step and %Mend statement ends definition of this macro.
- The “%Set_DD_MM_YY;” statement invokes this macro and returns &Mo_Yr.
- The “%OutFile(&Mo_Yr);” automatically generates / updates the OutFile_&Mo_Yr..CSV file on any specific day.

3. Generating Month-By-Month Trends

Up to this point, the author developed the SAS code and asked SAS to run that code automatically. It is now time to illustrate a technique for asking SAS to

1. Develop SAS code and then
2. Run that code

Suppose the user has values of a quality metric, say QM, for a Fab parameter (Technology, Logpoint, Equipment or a combination of these) for a generic Mo_Yr and wishes to develop a spreadsheet showing quality metrics for each value of that parameter month by month for judging Fab quality progress trends. It might be necessary to do this each new month as the data becomes available without recompiling the code. For instance, if Machines 1 through 50 were used in 03_2004; Machines 1 through 30, and 36 through 60 were used in 04_2004; Machines 11 through 60 were used in 05_2004, etc.; and the user wishes to obtain quality trend for each machine in each month when that machine was used. It is necessary to compute in trends in 03_2004, 04_2004, 05_2004, etc. Note that there is at least one additional column each new month.

Whereas SQL can deal with variable number of rows, it cannot deal with an indeterminate number of columns. Thus, one cannot have ONE source code to generate trends. In this case, it is extremely useful to ask SAS to develop the necessary code and then ask SAS to run that code. Indeed automation is not possible unless the developer resorts to this approach. Figure 4 shows the mechanism.

4.1 Notes on Figure 4

- The macro definition starts with the statement,

  %Macro Trends(Parm, Label, Mo_Yr);
- The first “Data _Null_” step determines the number of consecutive months for which trends are to be computed. The second Data _Null_ step sets up appropriate number of observations in each month. The starting month is fixed at 03_2004.
- To follow the logic, assume that &Mo_Yr = 05_2004, &Parm = Tech and &Label = Technology.
- Mon = two-character literal in &Mo_Yr starting at position = 1.

  Thus, Mon = ‘05’. Mo = numeric value of Mon = 5.
- Similarly, Yr = numeric value of the 4-character literal starting at character 4 = 2004.
- Months = 12 * (Yr – 2004) + Mo – 2 = 12*0 + 5 – 2 = 3. In other words, every day of May 2004, there will be 3 columns in the final table of QM Trends.

Figure 4: Ask SAS to Develop SAS Code and Then Run It

```
%Macro Trends(Parm, Label, Mo_Yr);
  Data _Null_;
    Mon = Substr(Symget('Mo_Yr'), 1, 2);
    Mo = Abs(Mon);
    Yr = Abs(Substr(Symget('Mo_Yr'), 4, 4));
    Months = 12*(Yr–2004)+ Mo – 2;
    Call Symput('Months', Left(Months));
    Call Symput('Mon', Trim(Mon));
  Run;
  Data _Null_;
    File "/home/account/library/Trends_SQL.SAS";
    Format next $2.  c_yr $4.;
    Put 'Proc SQL;';
    Put "Create table QM.&parm._all as";
    Put 'Select *';
    Put "From (Select coalesce (a.&parm, b.&parm) as &parm";
    Put "  label=" "'&label',";
    Put '      a.QM_03_2004, ';
    %Do I = 1 %To &Months-2;
      Next=&i+3;
      Pyr=Int(Abs(next-1)/12)+2;
      Next=Mod(next,12);
      If next=0 then next=12;
      If Abs(Next) < 10 Then Next='0'||Left(Abs(Next));
      C_yr=Left(pyr);
      Var='a.QM_'||Trim(Next)||'_'||Trim(c_yr);
      Put '     ' Var ',';
    %End;
    Put "    b.QM as QM_&Mo_Yr format=8.3";
    Put "  From QM.&parm._all as a";
    Put "     Full Join QM.&parm._&Mo_Yr as b";
    Put "     On a.&parm=b.&parm)";
    Put "  Order by &parm;";
    Put 'Quit;';
  Run;
  %Include "/home/account/library/Trends_SQL.sas";
%Mend;
```

- Store macro variables: &Months = 3, &Year = '2004' and &Mon = '05'.
- The second Data step will generate the SAS source file, Trends_SQL.SAS. Each Put statement creates one line in this source file. If the contents are within single quotes, then the resulting source line will be a verbatim copy. If contents are within double quotes, the source line will be a combination of verbatim contents augmented by

resolved values of macro variables, if any. If SAS variable information is outside quotes, then the source line will contain the resolved values of that SAS variable. Thus, the fist six Put statements will generate the following lines in the file Trends_SQL.SAS.

```
Proc SQL;
Create Table QM.Tech_All as
Select *
From (Select Coalesce (a.Tech, b.Tech) as Tech
     Label = 'Technology',
     a.QM_03_2004,
```

- At this stage SAS goes through a Macro Do Loop for &I = 1 %to 3–2 = 1, i.e., in this case, %Do is performed once leading to Next =&I + 3 = 4 and hence Next = '04'. In this case,

```
Var = 'a.QM_04_2004
```

- Thus the first Put statement after %Do loop results in the next Source line

```
a.QM_04_2004,
```

- The remaining 4 Put statements lead to the following source lines

```
b.QM as QM_05_2004
From QM.Tech_All as a
Full Join QM.Tech_05_2004 as b
On a.Tech = b.Tech)
Order by Tech;
Quit;
```

- Here is the complete source code for clarity.

```
Proc SQL;
Create Table QM.Tech_All as
Select *
From (Select Coalese (a.Tech, b.Tech)
      as Tech Label = 'Technology',
         a.QM_03_2004,
         a.QM_04_2004,
         b.QM as QM_05_2004
      From QM.Tech_All as a
      Full Join QM.Tech_05_2004 as b
      On a.Tech = b.Tech)
Order by Tech;
Quit;
```

- The "Run;" statement completes the second Data step and the %Include statement runs the resulting code and generates the QM.Tech_All dataset.
- This technique requires satisfaction of an initial condition, namely, one must copy QM.Tech_03_2004 into QM.Tech_All. At the start, it will have only one column. The source code made on April 1, 2004 updates QM.Tech_All using April 1 data. It will have two columns throughout April 2004. The source code made on May 1, 2004 will update QM.Tech_All using May 1 data. It will have three columns throughout May 2004; and so on.
- Once begun, all updates are automatic.

4.2 SQL Notes

- The **Coalesce** qualifier in SAS SQL operates as follows. For a specific observation, if **a.**Tech = **b.**Tech, then the **a.**Tech value is preserved and no new observation is added. On the other hand, if **b.**Tech is different from **a.**Tech, then a new observation is inserted in the resulting data set and it receives its value from table **b.**
- The **Full Join** qualifier uses data from both sets, **a** as well as **b**.
- The **On** qualifier matches observations from data sets **a** as well as **b** as long as there is a match.
- The **Order by** qualifier leads to sorting by the values of **Tech**.
- Since SQL was developed by an organization external to SAS, there are some differences in notations and conventions used by SAS and those used by SQL. For instance, an SQL Table is a SAS data Set; and a SAS Variable is an SQL Column. SAS does not use a comma ',' to separate variables in a list whereas SQL insists on using comma to separate columns in a list. One must exercise caution when using SAS and SQL in the same code.

5. Generating a .CSV File for Trends

The preceding section illustrated a technique for automatically developing month-by-month trends. The final result of that code was a SAS data set. The decision-makers who do not have access to SAS on their PC need a .CSV file to view those trends and that file must also be generated automatically. The next example will illustrate the necessary code for generating a .CSV file from the Trends data set. See Figure 5.

5.1 Notes on Figure 5

Rationale for asking SAS to develop SAS code has been discussed in Section 4. Figure 5 shows a Data _Null_ statement **within another Data _Null_ statement**. The outer Data _Null_ statement generates the SAS code that contains the inner Data _Null_ statement and then this inner Data _Null_ statement outputs the .CSV file. The details of how this works will test the reader's concentration powers. For the sake of brevity, those details will not be shown here. Note also that the two comments enclosed within /* and */ refer to the code that has been already displayed in Figure 4. It is not reproduced here in order to keep Figure 5 in the bounds of a single page.

Figure 6 shows the SAS code generated by the Macro in Figure 5 when &Mo_Yr = 05_2004, &Parm = Tech and &Label = Technology.

Figure 5: Automatic Generation of a .CSV File

```
%Macro Trends_csv(Parm, label, mo_yr);
  /* See Figure 4 for code to obtain Month, Mo, Yr, etc. */
  Data _Null_;
    File "/home/account/library /Trends_csv. SAS";
    Format next $2.  c_yr $4.;
    Put 'Data _Null_;';
    Put '  Set  QM.&parm._ALL end=EFIEOD;';
    Put '  %Let _EFIREC_ = 0;';
    Put '  File "$QM_TRENDS/&label..csv "' ;
    Put '       delimiter="," DSD DROPOVER lrecl=512;';
    Put '  If _N_ = 1 Then ';
    Put '    Do;';
    Put '       Put  "&label" "," "QM_03_02" "," ';
    /* See Figure 4 for Macro Do Loop for QM Columns
       Month-by-Month */
    C_yr=Symget('Yr');
    Var='QM_'||Trim(Symget('mon'))||'_'||Trim(c_yr);
    Put '              "' var '";' ;
    Put '   End;';
    Put '   Do;';
    Put '      EFIOUT + 1;';
    Put '      Put &parm @;';
    Put '      Put QM_03_02 @;';
    %Do I = 1 %To &Months-2;
        var='QM_'||Trim(next)||'_'||Trim(c_yr);
        Put   '       Put ' var ' @;';
    %End;
     c_yr=Left(yr)
    var='QM_'||Trim(Symget('mon'))||'_'||Trim(c_yr);
    Put '         Put ' var ';';
    Put ' End;';
    Put " If EFIEOD Then Call Symput('_EFIREC_',EFIOUT); ";
    Put 'Run;';
  Run;
  %Include "/home/account/library/Trends_csv. SAS";
%Mend;
```

Figure 6. SAS Code Generated By Macro Trends_csv

```
Data _Null_;
  Set QM.&parm._ALL End=EFIEOD;
  %Let _EFIREC_ = 0;
  File "/home/account/library/&label..csv " Delimiter=","
      DSD DROPOVER Lrecl=512;
  If _N_ = 1
  Then Do;
    Put  "&label" "," "QM_03_02" ","
       "QM_04_02 " ","
       "QM_05_02 ";
  End;
  Do;
    EFIOUT + 1;
    Put &parm @;
    Put QM_03_02 @;
    Put QM_04_02  @;
    Put QM_05_02 ;
  End;
  If EFIEOD Then Call Symput('_EFIREC_',EFIOUT);
Run;
```

6. Generating HTML Code

This application illustrates one other use of the “Data _Null_” constructs. In the Sherman facility of Texas Instruments Inc, many different semiconductor devices are combined into a number of process groups. All devices in a specific group are subject to the same electrical tests and for each such test, have identical specification limits. Consequently, when displaying the resulting test information on the corporate web page, there is no need to show information about each device on its own. Such information needs to be shown by device groups only. On the other hand, individual engineers or managers might not remember the specific group membership of every device.

The SAS code in Figure 7 shows the list of all devices on a single web page. When a reader clicks on a specific device name, it opens a new web page listing the links to the information about all electrical tests available for the device group to which this device belongs. (The user than then click on that link to get the test-specific information, for instance, a corresponding chart in the .GIF format. Proc Gplot was used to generate each chart and place it in a .GIF file.)

6.1 Notes on Figure 7.

It is necessary to generate a file in the HTML format. This is done using three distinct “Data _Null_” steps.

Figure 7: SAS Code to Generate HTML

```
Data _Null_;
  File '/home/account/library/Device_Group.htm';
  Put '<head>';
  Put   '<style>';
  Put '* {font-family: Arial; font-size: 10pt;
      text-decoration: none; } ';   /* Liberty */
  Put   '</style>';
  Put '</head>';
  Put '<center>';
  Put   '<div> <h1> ETEST DEVICES </h1> </div> <br>';
  Put '<table border="0" cellpadding="4">';
  Put "  ";
Run
;
Data _Null_;
  file '/home/account/library/Device_Group.htm' mod;
  set fmt.dev_group nobs=nobs;
  if mod(_n_,10)=1
      then  Put '<tr>';
  where='file://\\drive\account\Dir\_'||trim(group);
  Put '<td> <center> <a href="' where +(-1) '"> '
  device +(-1)'</a> </center> </td>';     /* Liberty */
  if mod(_N_,10)=0  or _n_=nobs
      then do; Put '</tr>';   Put "  ";  end;
Run
;
Data _Null_;
  file '/home/account/library/Device_Group.htm' mod;
  Put '</table>';
  Put '</center>';
Run;
```

- The first Data _Null_ step starts a file, Device_Group.htm and stores the HTML Header information in that file. As usual, the Put statements have bee used to develop the HTML **header** information such as: <head> <style> `{font-family: Arial; font-size: 10pt; text-decoration: none;` are the information items in the **header.** The right brace,}, closes the left brace,{, and `</style>` closes `<style>`. It was not possible to fit the <Style> on one line in Figure 7. Hence the /* Liberty */ comment.
- The </head> closes the <head> statement.
- The next three statements instruct HTML to place the words `ETEST DEVICES` in the center of a new line. They also instruct HTML to use 4-pixels padding between words.
- This completes Data _Null_ step 1.
- When this Data step is run, it results in the following HTML statements.

```
<head>
<style>
* {font-family: Arial; font-size: 10pt; text-decoration: none;}
</style>
</head>
<center>
<div> <h1> ETEST DEVICES </h1> </div> <br>
<table border="0" cellpadding="4">
```

- The second Data _Null_ step starts by telling SAS to modify the above HTML file by adding the statements to be developed next.
- One asks SAS to open the data set FMT.DEV_GROUP and store the Number of observations in a counter, NOBS. The data set .DEV_GROUP contains two items per line: the Device and its Group.
- The developer wishes to list 10 devices (hence the MOD function) on each line – unless it is the last line of the HTML file – and in that case, it will list all of the leftover devices on that line of the file:

  \\drive\account\Dir_(group)

  This file is in the DOS format as our users will access this web page from their PC's using Internet Explorer. The (group) notation is used to indicate that the name of the Device Group will be read from the current observation.

- The device names are centered in their columns and there is to be a blank line after every 10 lines of HTML. The HTML for initial 10 devices is shown next. (Each HTML line in the Landscape orientation needs two lines in the Portrait orientation.)

```
<tr>
<td> <center> <a href="file://\\ drive\account\Dir\_EPIC1ZS">
      CBTDN16211EIN</a> </center> </td>
<td> <center> <a href="file://\\ drive\account\Dir\_EPIC1ZS">
      CBTDN3306EIN</a> </center> </td>
<td> <center> <a href="file://\\drive\account\Dir\_EPIC1ZS">
      CBTDN3384DIN</a> </center> </td>
<td> <center> <a href="file://\\ drive\account\Dir\_EPIC1ZS">
      CBTDN3861EIN</a> </center> </td>
<td> <center> <a href="file://\\ drive\account\Dir\_EPIC1ZS">
      CBTHN16211EIN</a> </center> </td>
<td> <center> <a href="file://\\ drive\account\Dir\_EPIC1ZS">
      CBTKN16245IN</a> </center> </td>
<td> <center> <a href="file://\\ drive\account\Dir\_EPIC1ZS">
      CBTKN6800FIN</a> </center> </td>
<td> <center> <a href="file://\\ drive\account\Dir\_EPIC1ZS">
      CBTN16209EIN</a> </center> </td>
<td> <center> <a href="file://\\ drive\account\Dir\_EPIC1ZS">
      CBTN16211DIN</a> </center> </td>
<td> <center> <a href="file://\\ drive\account\Dir\_EPIC1ZS">
      CBTN16212DIN</a> </center> </td>
</tr>
```

- The third Data _Null_ step is used to close this HTML table.

7. Epilogue

This paper has necessarily focused on the mechanics of automatically generating statistical reports and graphs.

7.1 Other Applications

There are numerous other applications of this technology. Three are listed next.

- Dynamically determine the number of observations in a data set. A generic example follows.

```
Data _Null_;
  If 0 Then Set Name Point = _N_ Nobs = Count;
  Call Symput('Total', Left(Count));
  Stop;
Run;
```

The macro variable &Total can be used to make run-time decisions such as

- Whether to attempt to open the data set Name;
- Send an error message if &Total is smaller than a specified number, etc.
- In theory, wafers are to be probed using Tester A. At times, engineers might choose to use Tester B rather than A. The author has developed SAS code to identify such "Exceptions" and the associated changes in Wafer Probe times.

7.2 Streamlining Probe Times

When probing wafers from qualified devices, our wafer probe engineers routinely probe only a sample of dies on each wafer to assess its yield characteristics. If the sample yield is above a threshold, then the bad die identified in the sample are marked and the entire wafer is shipped to the assembly site. If the sample yield is below that threshold, then the entire wafer is probed, bad dies are marked and then that wafer is shipped to the assembly site. Naturally, probe time of these qualified devices has a bi-modal distribution and this affects management of their planned times in the wafer probe facility. The technology discussed in this paper was used to estimate probe time statistics (median, quartiles, and Tukey limits) within each 'sample good' and 'sample bad' sub-populations and these were weighted to derive usable probe planning times. Those plans have been successfully implemented in the Sherman facility and have been running without a glitch for a long period of time. The process of developing these planning numbers also identified and resolved a number of "improvement opportunities". Management was pleased with the success of this high-visibility project.

7.3 Process and Quality Improvements

The quality trends alluded to in this paper have identified a number of "improvement opportunities". An SPC Team has been empowered to investigate cost-effective ways of resolving these issues and the management has been actively reviewing these quality trends. Since these require changes in data entry practices, database management, specifications or

processes, the progress has not been as quick as in the probe time planning. There are plans to present a paper on the resultant quality improvements in the near future.

Acknowledgements

The author would like to thank Shawn Hughes, Mike Pinion, Roger Hale, and Michael Lake for their active support and many constructive suggestions during the development and testing of the software to support Wafer Probe Planning, Statistical Process Control (SPC) and E-Test Tracking projects. These projects could not have been successful without their active support. Thanks are also due to Linda Bailey and Mark Owen for making the Fab data accessible to SAS in the UNIX environment and to Eugene Gharis for getting this author started in HTML code for Internet Explorer.

Last but not the least; the author would like to thank Clyde Hepner, Sherman Wafer Fab and Plant Manager for his constant support and encouragement.

This paper summarizes some of the techniques used in support of the Sherman facility.

References

1. SAS Institute (eds.), 1990, *SAS Language: Reference*, SAS Institute Inc., Cary, NC
2. SAS Institute (eds.), 1989, *SAS Guide to the SQL Procedure*, SAS Institute Inc., Cary, NC
3. SAS Institute (eds.), 1994, *SAS Macro Facility: Tips & Techniques,* SAS Institute Inc., Cary, NC.
4. Mason, P., 1996, *In the Know: SAS Tips & Techniques,* SAS Institute Inc., Cary, NC.