Data Mining the Original Data Warehouse: Twenty-Five Years and a Million Lines of SAS Later

H. W. "Barry" Merrill, PhD
President-Programmer
Merrill Consultants
Dallas, TEXAS, USA

The author of MXG Software provides a historical perspective of how and why the SAS System became the pervasive tool for managing and mining of the original data warehouse, the Performance Data Base built from SMF data. The architecture of the MXG implementation is described to show how MXG, currently 911,749 lines of SAS code in 3,011 files (members), executes under MVS, VM, UNIX, OS/2, Windows 95 or Windows NT to create 1,908 SAS tables (datasets) with 78,278 columns (variables) from the raw data records produced by 268 products, where the input volume ranges from only hundreds of megabytes to fifteen gigabytes per day; some CICS tables contain in excess of 130 million rows (observations).

Contents

1. Notre Dame - 1959 - WOW! from an IBM 610 digital computer.

As a Notre Dame sophomore in EE in September, 1959, my first EE lab experiment was to calculate the determinant of a 4x4 matrix. As the ancient Lab Instructor finished his instructions he said, "I have to read this. The IBM corporation has donated a Model 610 dig-it-al computer, located in room 240, and students can sign up for hour-long blocks." Putting down the sheet of paper, he said "those digital things will never last, but next year, as juniors, you can learn to use the Bendix G15 Analog Computer - that's how engineers solve real problems!"

I went to room 240, looked through the peep hole and saw a large grey box, a table with typewriter, and what I assumed to be a senior, and opened the door to enter. As the door unhinged, so did the student, shouting "Shut that door!" as he strode across the room to the door, flailing his arms. As he stepped out into the hall shouting "Didn't you read the damn sign?", he discovered his sign had fallen face down on the floor. Calming, he informed me that you must get the operator's attention so he could put the machine in "QUIESCE/STOP" (which took 5-10 seconds), and only then was it safe to shuffle in -- slowly. The vacuum tube machine was so heat sensitive that the air currents would cause computation to fail, requiring a program restart.

He pointed me to the IBM manuals and I began at page one. Several hours later, I had learned to punch paper tape and print them on the Selectric and decided to calculate the determinant on my new toy.

By Saturday, I had punched my program, printed it, and was now ready to run my first computer program. As I watched the paper tape whir through the reader, the addresses flickering on the nixie tubes; I crossed my arms and thought, "Wow, it is 1959, I am a sophomore in college and am running a real program on a digital computer." The paper tape came to the end, the printer came alive, and I received my first computer output, four characters: **WOW!**

It took until Sunday to find the senior, who found that I had sort of missed the difference between "program" and "data". The first punch in the tape was a control character that put the 610 in a scan mode, and in the fifth-from-end position there was a control character to print the tape as machine instructions. What had been printed were the code letters for the last four program instructions:

W	=	Carriage Return
O	=	Line Feed
W	=	Carriage Return
!	=	Print Accumulator!

(Two Carriage Returns were always used to ensure that the very slow print head was all the way left before print.)

I did finally get the determinant computed, and submitted the first EE lab problem that used a digital computer at Notre Dame, but I did nothing further with computers while there.

I dropped out of Notre Dame in 1962, joined the Navy, was in the Cuban blockade on a surface ship, then on a Diesel submarine, and then won a Navy scholarship that sent me back to college in EE at Purdue University in 1964.

2. Purdue, 1964-1967 - IBM 7090/7094 and IBM 360/44.

At Purdue, I took a one-hour Fortran II course, using a 7090/7094, and was hooked. I worked on Linear Programs to model power grids, got a job in the Tab department wiring plug boards for sorters, collators, and printers, implemented the Fast Fourier Transform from the original Cooley-Tukey paper, worked for the Laboratory for Agricultural Remote Sensing (pattern recognition of crops from spectral data which led to the Earth Resource Technology Satellite), built the ground-truth data for LARS agronomists, and set fire to our 360/44 Serial #2 (twice!) with a tight loop in the floating point divide unit that lacked a heat sink. I showed one PhD candidate in Psychology how pattern recognition and vector distance could be used to cluster petroleum engineers that found oil from those that did not, and coded Fortran programs to manipulate data to invoke the BIMD statistical subroutines for another. I finished my BSEE and MSEE in August, 1967, but the Navy needed nuclear submarine drivers, not programmers, so again I set computing aside for a second masters in Nuclear Propulsion and sightseeing in the Barent's Sea, until shore duty running the airline to Guantanamo Bay, Cuba, where I taught calculus and ran the overseas extension for Old Dominion University.

3. State Farm Mutual Automobile Insurance Company, 1972-1976.

Leaving the Navy in 1972, my Psychologist friend, now working at State Farm Insurance in Bloomington, IL, suggested that I might find a home there.

Dave Vitek had gone to the Boole and Babbage User Group (BBUG, the predecessor of CMG) and decided that maybe, instead of trusting the IBM salesman as your capacity planner, State Farm could measure its own computers, and had funded a ten-person Measurement Unit for a feasibility study.

Steve Cullen had drafted an excellent attack plan to evaluate tools, and in short order we had Kommand/PACES for accounting, Software Monitors (SYSTEM LEAP and PROGRAM LEAP), Hardware Monitors (TESDATA XRAY), and Simulation (SAM).

Because Kommand was only for billing, Denny Maguire had started to write PL/1 programs to extract fields from SMF records, and I had revived an old Plot subroutine from LARS days, when I found this brief announcement in Datamation:

"*The Institute of Statistics at North Carolina State University announces the availability of the Statistical Analysis System, a package of 100,000 lines, one third each in Fortran, PL/1 and Assembler, that does printing, analysis and plotting of data."*

I wrote for information, and got a typical university document, with some pages dittoed, some pages typed, some printed, each on paper of a different color; but I immediately saw the power and simplicity of the INPUT statement for SMF data. However, in the list of supported data formats, there was no reference to Packed Decimal. You need to get only seven bytes into an SMF record to encounter a Packed Decimal field, so I called the Institute and asked Tony Barr, the author of the SAS compiler, about support.

"Well, we haven't got around to documenting it yet, but if you type in PD4. it will work jest fine" he said, so

I convinced State Farm to risk the 1972 purchase price of $100 for the SAS package.

Starting in 1964, Tony Barr and Dr. Jim Goodnight had collaborated to develop an ANOVA routine for the Department of Agriculture. Tony had been an IBM developer of the data base for the cold war's Distant Early Warning (DEW line) radar system, and Jim was a well known statistician.

Both recognized the weakness of the existing stat packages. They were only subroutines that had to be invoked by other programs that had to prepare and manage the data to be analyzed. By creating a language, a database, and the statistics, the Statistical Analysis System expanded well beyond the original ANOVA routine and had been tested at several Agricultural Experimental Stations and other universities.

The 1972 announcement was the first public release of the Statistical Analysis System, and in October, 1972, State Farm was the first real customer to install the SAS package from NCSU's Statistics Department.

Within days of receipt of SAS, we were extracting CPU time and PROGRAM name and K-Core-Hours to produce reports on resource consumption direct from SMF records, and, because SAS stores in floating point, we found that Kommand lost hours of CPU time because of truncation.

Presentations on the use of SAS software and the PDB were given to the Bloomington and Chicago chapters of the ACM and DPMA; the SAS data base was mentioned in my paper (on the use of the SAS data base to create simulation input for the System Analysis Machine directly from actual SMF data) presented at the 1973 SSCS (Symposium on the Simulation of Computer Systems) at NBS, and at a BOF session at the Seventh Annual Interface Symposium at Iowa State.

Many XRAY hardware monitor users became aware of State Farm's PDB through the Midwest TESDATA Users Group, which held its inaugural meeting in 1973 at State Farm. These presentations were only half technical; we also had to convince attendees that staffing of this new measurement concept was cost justified by the real dollar savings.

John Chapman had used an XRAY at Standard Oil and invited me to join SHARE's Computer Measurement and Evaluation (CME) project, and the PDB was described in a closed session of the CME project at SHARE 42 in Houston in March of 1974. The first open session presentation on the use of the SAS System to process SMF data was before an audience of over 750 (half of the attendees!) at SHARE 43 in August.

In 1987, SAS Institute published the 630-page red book

That session was split with an IBM presentation on their new Statistics Gathering Package, an FDP that selected a few fields from a few SMF records. IBM spoke first, then I showed what we had done with SAS at State Farm. One attendee stood and asked the IBM author of SGP, Bill Tetzlaff, "Now that you have seen SAS, is there any reason why you would still recommend your SGP product?"

Several hundred SHARE sites acquired SAS that fall as a result of this SHARE session!

In 1974, SAS added File 13, SAS.MERRILL, to their distribution tape with code examples for reading SMF data.

4. Sun Oil Company, 1976-1984.
In 1976, I completed my course work at the University of Illinois (65 miles each way on a CB500 Honda), and when State Farm decided not to rapidly migrate to the new MVS operating system, I left for Dallas and Sun Oil Company, where I demonstrated that the analysis of SMF with SAS was valid for VS2 as well.

In 1979 I wrote my dissertation, "A Comprehensive Approach to the Measurement of Large Scale Computer Systems" and received my PhD in EE from U of I. In 1979, Jane Helwig, director of publications at SAS Institute (which had become an independent company in 1975, marketing "the SAS System" instead of the "Statistical Analysis System") said users wanted a book and SAS code that showed how to measure computers, so we worked together on what was to be titled "The Analysis of SMF and RMF Data Using the SAS System".

Just before printing, Jane called to say that no one liked the name, and asked if my ego could handle the title "Merrill's Guide to Computer Performance Evaluation using the SAS System", which became a 395 page blue book, sold by SAS with a tape of sample programs for $395 in 1980.

By 1983, MVS/XA loomed with radical changes to SMF data, and many of the book's users were asking for a real software product, so in 1984, SAS published "Merrill's Expanded Guide to Computer Performance Evaluation Using the SAS System", a 835 page red book ($50), and SAS distributed the new MXG Software tape ($700) that was shipped with the then optional Merrill Consultant's "Support Subscription" agreement ($500 annually), and I left Sun Oil.

Judy, who had taught Business College and had been an executive with an apparel firm, said that she would run the business and I would write and support the software; she does and I do.

Figure 1 - Reading a Single SMF Record Type

"Merrill's Expanded Guide Supplement" and in 1991, Merrill Consultants replaced the old Support Subscription with a License Agreement and took over all distribution of MXG Software and MXG Books.

MXG Software has been installed at over 5,200 data centers in all states and 49 countries (although there are only about 3,000 licenses now, due to data center consolidations), and over 15,000 people bought the books.

5. Architecture of MXG Software SMF Processing - Single Record

So much for history. The design of MXG Software exploits many features of the SAS System, especially in the DATA steps that are used to convert raw SMF data into SAS tables (aka "datasets") that are stored in SAS data libraries (aka SAS "databases"). A simple SAS program to read the SMF file and decode type 0 (IPL) records is shown in Figure 1.

But to process more than one SMF record, a separate program for each record type would be needed, and the SMF file would have to be read once for each SMF record. Instead, for each record type, MXG creates one source member, VMAC0, that defines two "old-style" substitution MACROs, _VAR0 and _CDE0 with the code segments that are unique to each record:

```
  MACRO _VAR0  TYPE0  %
  MACRO _CDE0  Figure 1 code from LENGTH through
     END;   %
```

and one source member, VMACSMF, for the INFILE SMF code segment:

```
  MACRO _SMF   INFILE SMF ...;   %
```

so you can construct a SAS program that will read multiple SMF records in one pass of the SMF data using simple macro references to create SAS datasets from type 0, 6, 26, and 30, as shown in figure 2.

These old-style MACRO statements are used simply as shorthand; SAS will replace the macro name with its contents as SAS reads the source code. They were not replaced by the newer %MACRO facility, because MACRO can handle any text string, whereas %MACRO has real problems if the text has parentheses, and because %MACROS must be compiled, while MACROs are simply read and stored. All MXG substitution MACRO names start with an underscore.

The actual MACRO definitions for _VAR0 and _CDE0 in member VMAC0 can now be examined in detail in Figure 3 to see the SAS features used:

installing a new version of MXG is little more than

```
DATA TYPE0;
INFILE SMF;
LENGTH DEFAULT=4 IPLTIME 8;
FORMAT DOWNTM SMCAJWTM      TIME12.2
           IPLTIME                 DATETIME21.2;
INPUT  @1 MVSXAFLG                 PIB1.
       @2 ID                       PIB1.
       @3 SMFTIME             SMFSTAMP8.
       @11 SYSTEM              $EBCDIC4.
@;
IF ID=0 THEN DO;
    IPLTIME=SMFTIME;
    INPUT  @15 SMCAJWTM   PIB4.  /*SMF0JWT*/
           @19 SMFBUFF    PIB4.  /*SMF0BUF*/
           @23 VIRTSIZE   PIB4.  /*SMF0VST*/
           @27 SMCAOPT    PIB1.  /*SMF0OPT*/
           @28 REALSIZE   PIB4.  /*SMF0RST*/
    @;
    SMCAJWTM=60*SMCAJWTM;
    OUTPUT TYPE0;
    RETURN;
END;
```

Figure 2 - Reading Multiple SMF Record Types

```
%INCLUDE SOURCLIB(VMAC0,VMAC6,VMAC26,
    VMAC30,VMACSMF);
DATA
_VAR0 _VAR6 _VAR26 _VAR30;
_SMF
_CDE0 _CDE6 _CDE26 _CDE30;
```

Instead of the TYPE0 dataset name, MACRO _VAR0 contains the MACRO name _LTY0, the "Library" macro, that is defined in IMAC0, the "Installation Tailoring member" for the VMAC0 member. The default definition in IMAC0is MACRO _LTY0 TYPE0 %, to create the WORK.TYPE0 dataset; by using an externalized macro name in place of a hardcoded name, you can tailor IMAC0 to send the TYPE0 dataset to tape for a large dataset to save DASD space, or could rename TYPE0, without ever modifying the MXG Source Library. By concatenating a tailoring library that contains all of your changes ahead of the MXG Source Library:

```
//SOURCLIB   DD DSN=TAILORNG.SOURCLIB,DISP=SHR
//           DD DSN=MXG.SOURCLIB,DISP=SHR
```

four bytes for a DATETIME variable will truncate up to 255 seconds), and some accounting variables (like replacing the old MXG Version's Source Library with the new MXG Version's Source Library.

The KEEP= list inside the parentheses names the variables that are to be kept in dataset TYPE0. Without the dataset KEEP= operand, all variables defined in the DATA step would be kept. At the end of the KEEP= list is the _KTY0 token, defined in IMAC0, which defaults to null.

If you wish to create new variables NEWVAR1 and NEWVAR2 in dataset TYPE0, you would define

```
    MACRO _KTY0 NEWVAR1 NEWVAR2 %
```

to add those variables to KEEP= list. Moreover, if you want to drop variables that you don't need (to reduce the stored size of TYPE0), for example OPTDSETS and OPTVOL, you would define

```
    MACRO _KTY0 DROP= OPTDSETS OPTVOL %,
```

and those variables would not be kept in dataset TYPE0, because the DROP= list overrides the KEEP= list! You can also use the KTY0 macro name to add any dataset option (for example, COMPRESS=YES) for the TYPE0 dataset.

When variables are listed in a KEEP= list but are not created, those variables are not kept. This is exploited in CICS type 110 processing, where optional segments (DL/I, DBCNTL) may or may not exist. MXG names those variables in the KEEP= list, but the optional processing code (in IMACICDL,IMACICDB) is commented out, so SAS never sees the creation of those variables, and they are not kept. In this way, only one member, the optional IMACs, needs to be tailored to create them.

Inside the _VAR0 definition, the dataset LABEL= operand describes the dataset (that label is visible through PROC CONTENTS), and the KEEP= list is commented when new variables are added by a new release. Variables in the KEEP= list are listed alphabetically and aligned for ease in reading, but that ordering has no actual effect on the built dataset. Instead, by making the first SAS statement inside the_CDE0 macro to be the LABEL statement, and by listing variable names alphabetically, the dataset is created with variables in alphabetical order, so PROC PRINTs and PROC MEANS will show the variables in order, which is very useful when the dataset has hundreds of variables.

The LENGTH statement follows the LABEL statement and always contains DEFAULT=4, causing SAS to store numerics in only 4 floating point bytes (SAS default is 8 bytes) halving the DASD storage required. Four bytes floating point will store exact integers up to 16,777,216 and will keep seven significant digits, quite sufficient for most numerics. However, variables that contain DATETIME stamps require 8 bytes (using only Service Units) are also kept in 8 bytes (which will store 16 significant digits and integers as large as 72,057,594,037,927,936).

The FORMAT statement assigns the display format for variables, but does not affect the internal value of the SAS variable. Time variables are stored internally as seconds and fractions; most SMF durations have resolution of .01 second so they are FORMAT as TIME12.2 (999:59:59.99). DATETIME variables are stored internally as the number of seconds after Jan 1, 1960, the SAS epoch, and are FORMAT as DATETIME21.2 (15AUG2019:12:00:00, the startime of the third Woodstock) and display the four-digit year. Date variables are stored internally as the number of days since the SAS epoch and are FORMAT as DATE9. (15AUG2019).

Although FORMATs are assigned during creation of the variable, you can always override these formats in your reports with your own FORMAT statement. This is especially true of the DATETIME format. If you want to count IPLs by Date and Hour, you do not have to use DATEPART( ) and HOUR( ) functions to create new variables in a new dataset; instead, you can directly process the MXG dataset in your report program and shorten the format length:

```
    PROC FREQ DATA=TYPE0;
    TABLES IPLTIME;
    FORMAT IPLTIME DATETIME10.;
```

To count by date, use:

```
    FORMAT IPLTIME DATETIME7.;
```

Because of this SAS feature, MXG datasets always have one DATETIME variable instead of two separate Date and Time variables.

Following the FORMAT statement, variables LENGTH and OFFSMF are tested to detect invalid type 0 records (SYSPROGs who create SMF record often have unexpectedly created a record with ID=0). LENGTH and OFFSMF are created in MACRO _SMF, a portion of which is shown in Figure 4. The leading semicolon before INFILE is needed to terminate the DATA statement that will precede it (since the _VARxxxx tokens cannot end with a semicolon).

The OFFSMF test is initialization logic; once OFFSMF has been set, the RETAIN statement will keep that value. The JFCB=SMFJFCB operand on the INFILE puts the SMF Job File Control Block in variable SMFJFCB; if the fifth bit of the 100th byte of the JFCB is on, your //SMF DD points to an undumped VSAM SMF file. SMF records in the VSAM SMF file have four

Figure 3 - VMAC0

```
%INCLUDE SOURCLIB(IMAC0);  /* DEFINES _LTY0 AND _KTY0 */
MACRO _VAR0
_LTY0                                    /* TYPE0 */
        (LABEL='TYPE 0 IPL SMF'
        KEEP=DOWNTM IPLTIME OPTDSETS OPTVOL REALSIZE REC SMCAJWTM SMFBUFF  SYSTEM VIRTSIZE ZDATE
        /* ADDED BY MVS/ESA 5.1 */
        PRODUCT  SYSNAME   SYSPLEX
        _KTY0
        )
%
MACRO _CDE0
IF ID=0 THEN DO;
        LABEL
                DOWNTM       ='ESTIMATED*SYSTEM*DOWNTIME'
                IPLTIME      ='SMF*RECORD*TIME STAMP'
                OPTDSETS     ='CREATE*DATA SET/DASD*RECORDS?'
                OPTVOL       ='CREATE*VOLUME*RECORDS(19/69)?'
                PRODUCT      ='MVS*PRODUCT*NAME'
                REALSIZE     ='REAL*MEMORY*SIZE(KBYTES)'
                REC          ='TEMPORARY*SCRATCHES*(PERM/ALL)?'
                SMCAJWTM     ='JOB WAIT*(ABEND 522)*LIMIT'
                SMFBUFF      ='SMF*BUFFER*SIZE(BYTES)'
                SYSNAME      ='SYSNAME*PARAMETER*FROM IEASYSXX'
                SYSPLEX      ='SYSPLEX*NAME FROM*COUPLEXX'
                SYSTEM       ='SYSTEM*ID'
                VIRTSIZE     ='VIRTUAL*MEMORY*SIZE'
        ;
        LENGTH DEFAULT=4 IPLTIME 8;
        FORMAT DOWNTM   SMCAJWTM       TIME12.2
                        IPLTIME                 DATETIME21.2
                        ZDATE                   DATE9.
        ;
        IF LENGTH-OFFSMF NE 31 AND LENGTH-OFFSMF NE 56 THEN DO;
                NBAD0+1;
                IF NBAD0 LE 3 THEN
                        PUT     '***VMAC0.ERROR. INVALID TYPE 0 RECORD DETECTED. ' LENGTH= /
                                '   BUT TRUE IPL RECORD SHOULD BE EITHER 31 OR 56 BYTES '
                                '               RECORD IS DELETED ' _N_= SYSTEM= /       +16 SMFTIME=;
                DELETE;
        END;
        IPLTIME=SMFTIME;
        INPUT   @15+OFFSMF SMCAJWTM  &PIB.4.   /*SMF0JWT*/
                @19+OFFSMF SMFBUFF   &PIB.4.   /*SMF0BUF*/
                @23+OFFSMF VIRTSIZE  &PIB.4.   /*SMF0VST*/
                @27+OFFSMF SMCAOPT   &PIB.1.   /*SMF0OPT*/
                @28+OFFSMF REALSIZE  &PIB.4.   /*SMF0RST*/
        @;
        IF LENGTH-OFFSMF GE 56 THEN
        INPUT   @32+OFFSMF                       +1  /*RESERVED*/
                @33+OFFSMF PRODUCT     $EBCDIC8. /*MVS*PRODUCT*NAME*/
                @41+OFFSMF SYSNAME     $EBCDIC8. /*SYSNAME*PARAMETER*FROM IEASYSXX*/
                @49+OFFSMF SYSPLEX     $EBCDIC8. /*SYSPLEX*NAME FROM*COUPLEXX*/
        @;
        IF SYSTEM=PREVSYS AND IPLTIME GE PREVTIME THEN DOWNTM=IPLTIME-PREVTIME;
        OPTDSETS='NO ';
        IF SMCAOPT='...1....'B THEN OPTDSETS='YES';
        OPTVOL='NO ';
        IF SMCAOPT='...01...'B THEN OPTVOL='YES';
        REC='PERM';
        IF SMCAOPT='......1.'B THEN REC='ALL';
        SMCAJWTM=60*SMCAJWTM;
        %%INCLUDE SOURCLIB(EXTY0);  /* _LTY0 OUTPUTS TYPE0 */
        RETURN;
END;  /* END CDE0 */
%
```

Figure 4 - VMACSMF

```
MACRO SMF
 ;INFILE SMF STOPOVER LENGTH=LENGTH
    COL=COL START=BEGINCPY END=EOFSMF
    RECFM=VBS LRECL=32760 JFCB=SMFJFCB;
 RETAIN OFFSMF PREVID PREVSYS;
 IF OFFSMF=. THEN DO;
    IF SUBSTR(SMFJFCB,100,1)='....1...'B
         THEN OFFSMF=4;
    ELSE OFFSMF=0;
    %%INCLUDE SOURCLIB(IMACZDAT);
 END;
 INPUT@1+OFFSMF MVSXAFLG     &PIB.1.
         @2+OFFSMF ID                &PIB.1.
         @3+OFFSMF SMFTIME SMFSTAMP8.
         @11+OFFSMF SYSTEM     $EBCDIC4.
    @;
 %%INCLUDE SOURCLIB(IMACFILE);
%
```

bytes that are not present in the dumped SMF BSAM records; by setting OFFSMF=4 for VSAM and using +OFFSMF in the INPUT statement, those extra bytes are skipped, so you can transparently read either dumped BSAM or un-dumped VSAM SMF data.

IMACZDAT externalizes the code to set variable ZDATE (Zee Date Zee obs was created) so it can be changed in case of a rerun. The SMF header's four always-present variables are INPUT, and exit member IMACFILE is called. IMACFILE can be used to delete or select SMF records by time, ID, or system, to write selected raw SMF records to a flat file, etc.

Returning to the _CDE0 section, variable SMFTIME that was INPUT in the _SMF macro is stored into IPLTIME, and then the variables unique to the type 0 record are INPUT. SAS provides a wide range of SMF INFORMATS (SMFSTAMP, TODSTAMP and RMFSTAMP) that convert data into SAS datetime values; these unique INFORMATs work the same on all SAS platforms. However, INFORMATs IB, PIB, PD, RB, and NUM under MVS must be changed to S370FIB, S370FPIB, S370FPD, S370FRB, and S370FF when SAS is executed on ASCII platforms. For these INFORMATs, MXG uses a macro variable name (&PIB) in its source code, and member VMXGINIT (invoked by the CONFIG member at startup) executes either %LET PIB=S370FPIB or %LET PIB=PIB, depending on the execution platform, for transparent execution anywhere!

While numeric variables can be used for fields containing hex values, using HEX format for display, MXG now INPUTs hex fields into character variables with INFORMAT $CHAR and assign a $HEX FORMAT, because a one byte character variable stores in one byte, while a one-byte numeric needs two storage bytes on MVS and three bytes on ASCII platforms, and a four-byte numeric must be stored in five bytes to see all of the bits.

MXG detects when IBM has added new data to a record, to make MXG fully backwards compatible with all levels of MVS. The test for LENGTH-OFFSMF detects the new data added by MVS/ESA 5.1. While bits in MVSXAFLG might have been used to identify the MVS version that wrote the record, it is usually safer to test the record length rather than a version indicator, because developers must set the length correctly, but do not always update version numbers!

Although not shown in the _SMF macro in Figure 4, variables PREVSYS and PREVTIME contain the SYSTEM and SMFTIME from the immediately preceding SMF record, so that variable DOWNTM, an estimate of the outage, can be calculated for the TYPE0 (IPL) dataset. Then _CDE0 creates variables, converts SMCAJWTM to seconds, and the TYPE0 Data Set Exit member EXTY0 is INCLUDEd. That member contains comments and the SAS statement:

```
 OUTPUT _LTY0;
```

to externalize the OUTPUT of the TYPE0 dataset. In the Data Set Exit you can create new variables to be output, or you can add logic to conditionally execute the OUTPUT statement and thereby output only certain observations. The exit is taken after all variables have been created, so any criteria can be used for selection. The Data Set Exit member plus the _L and _K macro definitions in the IMAC member allow complete user tailoring of the contents of all MXG datasets.

6. Architecture of Building the Performance Data Base - BUILDPDB

I coined the term PDB, for the Performance Data Base, while at State Farm in the fall of 1972, when I began to regularly use my SAS program to process weekly SMF data records (from OS/360 MVT Release 20.6!). I recall a comment that we would need to build a warehouse to store the SMF data tapes, and a reply that we would let SAS manage the warehouse! The original PDB used only type 0, 1, 4, 5, 6, and 12 SMF records, and the weekly PDB, A.PERF155(0), was built on a mountable 3330-1 device; six weekly PDBs would fit on one 3330 back then! I implemented the daily PDBs in 1973, at the request of Operations, who had come to like the weekly reports and wanted daily detail. All trending, capacity planning and serious analysis must use weekly data, rather than daily or monthly, to see the true growth, because holiday-containing weeks fall out of a weekly plot, while monthly data varies according to the number of work days (and when they fall in the week) and cannot be normalized safely.

The term PDB was in wide-spread use inside State Farm by 1973, when Mario Morino and Doug Denault came to Bloomington, to install the first copy of their new TSO/MON product. They arrived Monday morning, and with the help of Kathy Colbert and Steve Cullen, they had assembled and started the monitor by noon. They then began to compile their COBOL report programs, which were still compiling at 6pm when Kathy handed me the SMF format for the TSO/MON SMF record as they went out to supper. I wrote the SAS code to decode the new record, added it to the PDB job, and came in the next morning to find the new code was successful; when Mario and Doug showed up at 8am, I gave them a SAS plot of the number of TSO users versus time of day, and thus did Mario learn of the power of SAS! (Only late that second day did their programs produce any reports!)

The MXG BUILDPDB logic consists of five phases. In the first phase, the input SMF file is read and multiple SAS datasets are created in the //WORK data library. The simple macro references for this phase are:

```
DATA
_VAR0         _VAR0203  _VAR6      _VAR21
_VAR26J2      _VAR30    _VAR7072   _VAR71
_VAR73        _VAR74    _VAR75     _VAR77
_VAR78        _VAR89    _VAR110    _VARDB2
 _VARTMNT     _VARUSER
 _SMF
_CDE0         _CDE0203  _CDE6      _CDE21
_CDE26J2      _CDE30    _CDE7072   _CDE71
_CDE73        _CDE74    _CDE75     _CDE77
_CDE78        _CDE89    _CDE110    _CDEDB2
_CDETMNT      _CDEUSER;
RUN;
```

This DATA step creates 116 SAS datasets in the //WORK library and creates the CICSTRAN dataset (one observation per CICS transaction, from type 110) direct to tape (to minimize DASD space and yet capture each CICS transaction). The programs ASUMCICS and ASUMDB2A, executed after BUILDPDB, read the detail transaction datasets to create the summary datasets PDB.CICS and PDB.ASUMDB2A that are compact for CICS and DB2 response and resource measurement.

Four PDB exits, EXPDBINC, EXPDBVAR, EXPDBCDE, and EXPDBOUT, allow other SMF records to be added to the PDB in the one reading of the SMF file.

In the second phase of BUILDPDB, datasets TYPE0, TYPE0203, TYPE21, TYPE30_D, TYPE30_6, TYPE30OM, TYPE30MU,TYPETMNT, TYPETSWP and TYPETALO are SORTed from the //WORK data library into the //PDB data library. By creating PDB datasets in SORT order, report programs do not have to do a SORT; instead, they can use BY statements with the report procedures to save CPU time and I/O activity for reporting. The sort order is preserved into the WEEKLY and MONTHLY PDB libraries as well.

The third phase SORTs these RMF datasets into the //PDB data library:

```
TYPE70        TYPE70PR  TYPE71
TYPE72        TYPE72DL      TYPE72GO
TYPE72MN      TYPE72SC      TYPE73
TYPE73L       TYPE73P       TYPE74
TYPE74CF      TYPE74ST      TYPE74TD
TYPE75        TYPE77        TYPE78
TYPE78CF      TYPE78CU  TYPE78IO
TYPE78PA      TYPE78SP      TYPE78VS
```

and then invokes member RMFINTRV to read and MERGE these datasets:

```
TYPE70        TYPE71        TYPE72
TYPE72GO      TYPE73P       TYPE74
TYPE75        TYPE78
```

to create one observation per RMF interval in the PDB.RMFINTRV dataset. PDB.RMFINTRV contains the PCTCPUBY from type 70, and (by using member IMACWORK to map performance groups or service classes to workloads) type 72 resources are summed in Batch, TSO, CICS, etc workload variables, so uncaptured CPU time and capture ratio can be measured, and the paging swapping and I/O activity is contained in a single observation for each RMF interval, providing hour-by-hour capacity measurement data.

The fourth phase reads the 47 CICS statistics datasets that were written to the //WORK library and invokes member CICINTRV to summarize them into the PDB.CICINTRV (interval) and PDB.CICEODRV (shutdown) CICS datasets.

The fifth phase of BUILDPDB operates on the type 30 subtype 1, 4, and 5 records, the type 6, and type 26 records to create accounting, resource and activity data for Jobs, TSO, STC, APPC, and Open MVS address spaces. Records for incomplete jobs (i.e., executed but not printed, or still running when SMF was dumped, etc.) that were written yesterday to the PDB's //SPIN data library are merged in with today's new SMF data. The completed jobs are written to the PDB.JOBS (one observation per job, 223 variables), to the PDB.STEPS (one per step, 200 variables), and to the PDB.PRINT (one per print file, 51 variables) and job accounting fields are propagated into

PDB.STEPS and PDB.PRINT so that resource billing by account at the job, step, or program level can be done directly from the PDB. Today's still-incomplete job records are written to the //SPIN library for tomorrow's processing. The IMACSPIN member defines SPINCNT, the number of days BUILDPDB will SPIN records. While the PDB.JOBS, PDB.STEPS and PDB.PRINT data sets contain observations for completed jobs, the dataset PDB.SPUNJOBS describes all jobs in the //SPIN library, so all of yesterday's work can be analyzed. Holding the records in the //SPIN library until the job has purged produces a single picture of the job. If records are not SPUN, one day's PDB can have an obs with only the CPU time from the type 30s, another day's PDB can have an obs with only print lines from the type 6, and yet another day's PDB will have an obs with just Purge record data, and all three obs from this one job will have many missing values and an incomplete picture, although no resources are lost. The individual SPINxxxx data sets are copied from the SPIN library into the PDB library for backup purposes. You can always go back to the last successful run, copy the SPINxxxx datasets from that PDB into the SPIN library, and then restart after an error.

While dataset TYPE70PR (one obs for each PR/SM, MDF, or MLPF LCPUADDR in each LPAR) contains LPAR utilization, INCLUDing ASUM70PR after BUILDPDB summarizes TYPE70PR to creates the more usable PDB.ASUM70PR dataset with resources consumed by each LPAR and with total CPU busy for all LPARS.

7. Mining costs and tons of warehouse data dug up and delivered:
The first phase of BUILDPDB took 1 hour and 1 minute of elapsed time and 16 minutes of CPU time on an IBM 9021-952 to process two 3490 tapes with 596,814 SMF records totalling 2,314 Megabytes (2.3 Gigabytes) of raw SMF data from three MVS systems. The total BUILDPDB step plus the ASUM70PR, ASUMDB2A, ASUMCICS and ASUMJOBS summaries took 1 hour 55 minutes elapsed and 24 CPU minutes. The WORK file required only 256 Megabytes (324 cylinders of 3390).

This Performance Data Base PDB output data library contains 143 datasets totalling 3,040 Megabytes, but 2,290 of those Megabytes are in CICSTRAN for its 2,333,338 CICS transactions. The high-volume CICSTRAN dataset is written directly to tape, to archive each transaction, but using no DASD space. Then CICSTRAN's 2,290 MB are summarized into only 21 MB in the 214,088 observations of the PDB.CICS summary dataset by ASUMCICS (which invokes the VMXGSUM generic summarization %MACRO and took only 22 minutes elapsed and 3 and a half minutes of CPU for that summarization).

So the online DASD PDB required only 910MB (1150 cylinders on a 3390). Its contents are shown in Figure 5. Most of that volume is taken by only a few datasets: DB2ACCT (427MB for 266,513 DB2 plan executions, which could have been sent to tape like CICSTRAN), STEPS (83MB for 88,777 steps), ASUMDB2A (69MB, the summarized output from DB2ACCT), TYPE74 (48MB), JOBS (34MB for 31,472 job executions), TSOMSYST (30MB), TYPE30MU (22MB) and CICS (21MB), plus all other RMF data (24 MB) account for 760 MB. But many important datasets are less than one megabyte; the RMFINTRV summary dataset has 24 hours of each of the 3 system's 15-minute RMF interval data (288 obs) in only 452 Kilobytes (9 tracks) of DASD space!

And what about that 130 million observation CICSTRAN dataset? One massive site read that many CICS transactions from 68 SMF tapes (3490E compressed) to select 1500 CICS transactions. The job took 30 elapsed hours and 5 CPU hours!

8. Growth of the MXG Source Library
An Annual MXG Version is created in first quarter of each year, and throughout the year, interim Versions are created as needed to keep up with new products and changes to old records. The fourteenth MXG Annual Version will ship in early 1997, but MXG 14.06, the August, 1996 Current Version, was the 99th MXG Version created! Figure 6 lists the measurements of each of the Annual MXG Versions and MXG 14.06. Sometime in 1997, the MXG source library should exceed 1,000,000 source lines.

The original MXG 3420 Tape Reel contained 99 feet for it's 22,000 lines of source code!

9. SAS does not stand for Single Authored Software. Acknowledgements.
While I have designed all and written most of the MXG Software, many users have contributed code examples, and my three consultants who have ably tested each new release, have covered my technical calls while I am out teaching, and have personally contributed significantly to MXG, are hereby acknowledged specifically:

Chuck Hopf
Bruce Widlund
Freddie Arie

Overseas, where we are represented by the local SAS Office, there are also scores of dedicated SAS technicians who have provided local language and local time of day help with MXG queries.

Figure 5 - PDB Contents

Dataset	Obs	Vars	Len	Mbytes	Description
ASUM70OR	288	218	836		RMF PR/SM LPAR INTERVAL
ASUMDB2A	52110	323	1383	69	DB2ACCT INTERVAL SUMMARY
CICAUSS	4648	31	172	1	CICS AUTOINSTALL TERMINAL USS
CICAUTO	96	43	203		CICS AUTOINSTALL GLOBAL
CICCONMR	2110	37	183		CICS ISC/IRC MODE ENTRY
CICCONSR	2044	48	223		CICS ISC/IRC SYSTEM ENTRY
CICCONSS	90	27	139		CICS ISC CONNECTION - SECURITY
CICDQG	96	38	183		CICS TDQUEUE TRANSIENT DATA GLOBAL
CICDQR	1002	28	140		CICS TDQUEUE TRANSIENT DATA SPECIFIC
CICDS	96	87	367		CICS DISPATCHER, CPU BY TCB
CICDTB	96	21	115		CICS DYNAMIC TRANSACTION BACKOUT
CICEODRV	96	290	1180		CICS END OF DAY
CICFCR	2352	70	435	1	CICS FILE CONTROL
CICINTRV	0	290	1180		CICS INTERVALS
CICJCR	196	32	162		CICS JOURNAL CONTROL
CICLDG	576	43	208		CICS LOADER DOMAIN FOR PROGRAM
CICSLSRFR	688	25	135		CICS LSRPOOL FILE STATS EACH FILE
CICSLSRR	76	294	1220		CICS LSRPOOL POLL STATS EACH POOL
CICM	96	27	139		CICS MONITOR DOMAIN GLOBAL
CICPAUTO	96	22	119		CICS AUTOINSTALL PROGRAM
CICS	214088	23	105	21	CICS INTERVAL SUMMARY
CICSDG	96	21	115		CICS SYSTEM DUMP GLOBAL
CICSDR	496	24	131		CICS SYSTEM DUMP SPECIFIC
CICSEXCE	205	39	194		CICS EXCEPTIONS
CICSMD	15092	35	164	2	CICS STORAGE MANAGER
CICSMDSA	768	64	335		CICS STORAGE MANAGER DSA AND EDSA
CICSMT	384	30	146		CICS STORAGE MANAGER TASK SUBP
CICST	96	22	119		CICS STATISTICS DOMAIN GLOBAL
CICSTRAN	2133338	260	1126	2290	CICS TRANSACTIONS
CICTCR	12476	35	166	2	CICS TERMINAL CONTROL SPECIFIC
CICTDG	96	21	115		CICS TRANSACTION DUMP GLOBAL
CICTDR	1368	24	127		CICS TRANSACTION DUMP SPECIFIC
CICTM	96	53	243		CICS TABLE MANAGER GLOBAL
CICTSQ	96	57	259		CICS TSQUEUE TEMPORARY STORAGE
CICUSG	96	24	127		CICS USER DOMAIN STATISTICS
CICUSSRV	147	290	1180		CICS USS INTERVAL SUMMARY
CICVT	96	30	151		CICS VTAM GLOBAL
CICXMC	1050	37	183		CICS TRANSACTION MANAGER TCLASS
CICXMG	96	31	155		CICS TRANSACTION MANAGER GLOBAL
CICXMR	32216	21	168	5	CICS TRANSACTION MANAGER TRANSACTION
DB2ACCT	266513	354	1680	427	DB2 PLAN ACCOUNTING
DB2ACCTB	0	46	294		DB2 PLAN BUFFER POOLS
DB2ACCTG	11	39	39		DB2 PLAN GROUP BPOOLS
DB2ACCTP	0	79	458		DB2 PLAN PACKAGES
DB2GBPAT	0	23	105		DB2 GLOBAL BUFFERS
DB2GBPST	0	44	148		DB2 STAT GB POOLS
DB2STAT0	571	540	2130	1	DB2 STATS SUBTYPE0
DB2STAT1	571	510	1987	1	DB2 STATS SUBTYPE1
DB2STAT2	2296	24	111		DB2 STATS SUBTYPE2
DB2STATB	1150	80	348		DB2 STATS BUFFER POOLS

DB2STATR	357	54	256		DB2 REMOTES
DB2STATS	571	1037	4047	2	DB2 STATISTICS INTERVAL
DDSTATS	0	26	148		TYPE 30 DD SEGMENTS
IPLS	0	14	70		IPLS
JOBS	31472	223	121	34	JOBS/TSO/STC/APPC/OMVS
JOBSKED	194	19	98		JOB SCHEDULING CLASS SUMMARY
NJEPURGE	0	62	357		NJE PURGE EVENTS
PRINT	15322	51	363	5	TYPE 6 PRINT EVENTS
RMFINTRV	288	402	1608		RMF INTERVAL SUMMARY
RRTMPSE	718	67	301		ROSCOE ACCOUNTING
SMFINTRV	0	218	1078		SMF INTERVAL ACCOUNTING
STEPS	88777	200	989	83	STEPS FOR JOBS/TSO/STC/APPC/OMVS
TAPEMNTS	0	18	1		MXGTMNT TAPE MOUNT SUMMARY
TAPES	1862	28	124		TYPE 21 TAPE DISMOUNTS
TSOMCALL	9075	110	573	5	TSO.MON USER RESPONSE
TSOMSYST	39372	183	785	30	TSO/MON SYSTEM INTERVAL
TYPE0203	140	5	27		SMF DUMP HEADER/TRAILER
TYPE23	72	22	110		SMF INTERVAL STATISTICS
TYPE30MU	142337	24	165	22	MEASURED USAGE DATA
TYPE30_6	1392	148	636	1	SSTEM ASID INTERVALS
TYPE37	7941	102	1091	8	NETVIEW HARDWARE MONITOR EXTERNAL
TYPE70	288	382	1435		RMF CPU ACTIVITY
TYPE70PR	4800	34	121		RMF PR/SM LPAR ACTIVITY
TYPE71	288	307	1248		RMF PAGING ACTIVITY
TYPE72	15941	89	396	6	RMF PERFORMANCE GROUP
TYPE7204	0	73	321		RMF MONITOR III GOAL MODE
TYPE72DL	0	35	166		RMF GOAL MODE DELAY SAMPLES
TYPE72GO	0	151	797		RMF GOAL MODE
TYPE72MN	14699	81	345	5	RMF MONITOR III COMPAT
TPYE72SC	0	16	97		RMF SERVICE CLASS SERVED
TYPE73	47520	45	197	9	RMF CHANNEL PATH
TYPE74	118712	101	423	48	RMF DEVICE ACTIVITY
TTYPE74CA	0	149	483		RMF CRR CACHE CONTROLLER
TYPE74CF	34	188	1050		RMF COUPLING FACILITY
TYPE74OM	0	82	336		OPENMVS KERNEL ACTIVITY
TYPE74ST	1728	56	259		RMF CF STRUCTURE REQUESTS
TYPE74TD	288	11	61		RMF TAPE DRIVES ALLOCATED
TYPE75	1344	40	225		RMF PAGE DATASET ACTIVITY
TYPE77	9917	43	266	3	RMF ENQ ACTIVITY
TYPE78	0	29	128		RMF I/O QUEUEING
TYPE78CF	15080	32	131	2	RMF I.O CONFIGURATION
TYPE78CU	5155	23	110		RMF LCU CU-HDR QUEUEING
TYPE78IO	1152	23	107		RMF IOP QUEUE
TYPE78PA	0	102	569		RMF JOB-LEVEL SUBPOOLS
TYP78VS	288	445	2430		RMF VIRTUAL STORAGE
TYPE89	0	29	188		MEASURED USAGE INTERVAL
TYPETALO	0	42	286		MXG TAPE ALLOCATION MONITOR
TYPETMNT	0	16	82		MXG TAPE MOUNT MONITOR
TYPETSWP	0	7	33		MXG TAPE SWAP EVENT
TOTAL SIZE				3200	

Figure 6 - History of Mxg Versions and Releases

Version	Date	Members	Lines	Bytes	Mbytes	Pages	Datasets	Vars
14.06	20AUG96	3011	911,749	72,939,920	69.6	15,196	1908	78,278
13.13	20JAN96	2940	885,344	70,827,520	67.3	14,756	1868	76,219
12.12	20MAR95	2533	776,687	62,134,960	59.3	12,945	1573	66,221
11.11	26MAR94	2286	654,341	52,134,960	49.9	10.975	1360	55,168
10.10	15MAR93	1965	534,902	42,792,160	40.8	9,999	1195	47,296
9.9	1MAR92	1605	388,651	31,092,080	29.6	6,477	1093	41,332
8.8	8APR91	1367	200,000	24,000,000	22.8	5,000	872	30,420
7.7	15FEB90	1100	230,000	18,400,000	17.5	3,834	679	22,277
6.6	15JAN89	910	165,614	13,249,120	12.6	2,760	552	18,048
5.5	01FEB88	785	136,880	10,951,296	10.7	2,281	456	14,909
4.4	01MAR87	661	108,166	8,653,472	8.8	1,803	360	11,770
3.2	01FEB86	537	79,444	6,635,648	6.8	1,346	264	8.632
2.0	01FEB85	413	50,722	4,057,024	4.9	845	169	5,526
1.0	AUG84	289	22,000	1,760,000	3.0	367	73	2,387

But our real thanks for our success are to the dedicated MXG users, who have taken the time to learn those prerequisite skills of SAS, JCL, TSO, and PC technology, and who have waded through 1,497 pages in two Books, 730 pages in 30 Newsletters, and 2,994 Changes in 99 Releases to learn how to use MXG Software to measure and thereby improve the performance of their company's computer systems - they have made all of us look good!

Chapter 99

This chapter cites and thanks all contributors to MXG Software, who are hereby officially awarded the title of "Chapter 99 CodeSharks"! Named by Judith S. "99" Merrill, Vice President and Partner, in honor of the diligent sleuthing of the MXG code performed by each CodeShark, she also established our policy that credit should be given to every MXG user who made any contribution to MXG Software, whether they provided an entire program, or found a programming error, or offered a suggestion, or even just found a comma out of place in a comment!

Contributors with Nine or more Changes - Descending Ranking

Name	Total Changes
Chuck Hopf	130
Diane Eppestine	50
Norbert Korsche	45
Freddie Arie	40
Tom Parker	34
Joseph Faska	28
Pete Shepard	27
Siegfried Trantes	23
Don Deese	19
Bruce Widlund	18
Rodney L. Reisch	18
Susan Marshall	17
Tom Elbert	16
Jim Gilbert	15
Dan Kaberon	14
Neil Ervin	14
Waldemar Schneider	14
Rod Fernandez	12
Shaheen Pervaiz	12
Bob Rutledge	11
Dan Squillace	11
Dick Whiting	11
Don Friesen	11
George Scott	11
Ray Dickensheets	11
Barry Pieper	10
David Ehresman	10
Steve Glick	10
Chris Weston	9
Elliot Weitzman	9
Paul Walker	9
Wolfgang Vierling	9