CGMXG – Chuck’s Guide to Merrill’s Expanded Guide

Contents

[Chapter 1: Some History 2](#_Toc405381233)

[Chapter 2: The Structure of MXG 3](#_Toc405381234)

[Table1: MXG Naming Conventions 3](#_Toc405381235)

[Chapter 3: Installing MXG – zOS or ASCII? 7](#_Toc405381236)

[Chapter 3a: Installing MXG on zOS 7](#_Toc405381237)

[Method 1 – IEBUPDTE 7](#_Toc405381238)

[Method 2 – TERSE UNPACK 8](#_Toc405381239)

[Running PROC FORMAT on zOS 8](#_Toc405381240)

[Chapter 3b: Installing MXG on an ASCII platform 9](#_Toc405381241)

[Chapter 4: Customizing MXG 14](#_Toc405381242)

[IMACSHFT 14](#_Toc405381243)

[IMACKEEP 20](#_Toc405381244)

[UTILWORK 24](#_Toc405381245)

[ANALACTM 27](#_Toc405381246)

[Chapter 5: Assembler Modules 28](#_Toc405381247)

[ASMTAPEE – the MXG Tape Mount Monitor 28](#_Toc405381248)

[ASMRMFV – RMF III datasets 28](#_Toc405381249)

[EXITCICS – CICS/DB2 Decompression exit 29](#_Toc405381250)

[EXITMON6 – Landmark Monitor Decompression 29](#_Toc405381251)

[Chapter 6: Tailoring BUILDPDB in 1987 – the hard way 29](#_Toc405381252)

[EXPDBINC would have looked like: 29](#_Toc405381253)

[EXPDBVAR would be: 30](#_Toc405381254)

[EXPDBCDE would be: 30](#_Toc405381255)

[And finally EXPDBOUT: 30](#_Toc405381256)

[Using MACKEEP to override EXPDB members 30](#_Toc405381257)

[Chapter 7: Using UTILBLDP 31](#_Toc405381258)

[UTILBLDP Parameters 31](#_Toc405381259)

[Some examples of using UTILBLDP. 33](#_Toc405381260)

[Chapter 8: BLDSMPDB 35](#_Toc405381261)

[BLDSMPDB Parameters 36](#_Toc405381262)

[BLDSMPDB Examples 39](#_Toc405381263)

[Chapter 9: AUTOALOC\VMXGALOC 40](#_Toc405381264)

[VMXGALOC Parameters 40](#_Toc405381265)

[VMXGALOC Examples 41](#_Toc405381266)

[Chapter 10: How much space will I need? 42](#_Toc405381267)

[Chapter 11: Running BUILDPDB on zOS 43](#_Toc405381268)

[JCL to create a GDG 43](#_Toc405381269)

[JCL for DAILY Job with dumped SMF data 46](#_Toc405381270)

[JCL for Daily Job Reading LOGSTREAM 46](#_Toc405381271)

[JCL for Weekly Job 47](#_Toc405381272)

[JCL for Monthly Job - disk 49](#_Toc405381273)

[JCL for Monthly Job - Tape 50](#_Toc405381274)

[Chapter 12: Running BUILDPDB on ASCII 51](#_Toc405381275)

[Daily JOB on ASCII Platform 51](#_Toc405381276)

[Chapter 13: What about non-SMF sources? 55](#_Toc405381277)

[DCOLLECT JCL 55](#_Toc405381278)

[ASCII FILENAME Statements for DCOLLECT and TMC 56](#_Toc405381279)

[Read DCOLLECT data on zOS 56](#_Toc405381280)

[Read DCOLLECT data on ASCII 57](#_Toc405381281)

# Chapter 1: Some History

A long time ago in a state farm far far away – OK, in Illinois – Barry Merrill developed some SAS code to read and process SMF data. Eventually, this became Merrill’s Guide to Computer Performance Evaluation, his doctoral thesis, and a book/tape from SAS for what was the then exorbitant price of 395 dollars. Soon everyone with MVS (well almost everyone) had a copy since it was pretty much the only way to get anything meaningful out of the SMF data being generated by MVS/370.

In 1982 I took Barry’s course in CPE in Dallas. Later that year at CMG Barry received the Michelson award for his work and in one session asked if anyone that was migrating to MVS/XA would be interested in helping in the development of what would become MXG. I quickly volunteered but was (initially) declined as Barry had made arrangements with Texas Instruments to do the development there. When that arrangement failed, I got a call from Barry asking if we (Computer Language Research) were still interested and so it began - what is now a more than 40 year association with Merrill Consultants and MXG.

By the summer of 1983 MXG version 1 was ready for distribution on 3420 tape at 22,000 lines of SAS code that created 73 SAS datasets with 2387 variables. Today (July 2014) version 32.05 of MXG is now over 3 million lines of SAS code creating 5479 SAS datasets with 245,875 variables. There has been a lot of change and growth in 33 years! There are almost 800 times more datasets and 100 times more variables. A single member of the MXG SOURCLIB (ANALDB2R) is now over 3000 lines of code larger than release 1 of MXG.

Over time, MXG has evolved from something that required some SAS programming skills to one where the required skills to install and care for MXG are not as rigorous as they once were. But, the documentation on how to get from complex to relatively simple has to some extent fallen short. My intent here is to explain both the complex and the simplified answers and to provide some guidance into some of the more esoteric features.

# Chapter 2: The Structure of MXG

MXG is delivered in source code form. While there are some assembler programs included none of them are required for the successful execution of MXG unless the data being processed requires them (more on that later.) The structure of the members in the SOURCLIB are based on old technology – an MVS PDS (partitioned dataset) and card images. So, the names in the SOURCLIB are limited to 8 characters and the lines in each are limited to 72 bytes (the records are 80 bytes but 73-80 are reserved for line numbers in this structure and none of the MXG members are numbered.)

For the most part, building an MXG ‘program’ is simply a matter of putting the building blocks contained in the SOURCLIB together in the correct sequence. We will get into a lot of detail on how to do it. Both the hard ways and the simple ways. But first, it is important to understand the naming conventions in the SOURCLIB.

There is a pattern to the member names. Table 1 lists the prefixes to the member names and their purposes. This list is not all encompassing but the members left out are generally archaic or very seldom used.

Table1: MXG Naming Conventions

|  |  |  |
| --- | --- | --- |
| **Member Name** | **Function** | **User Modifiable** |
| AAAAAAAA | The first member in the library  |  |
| ACHAP\* | The chapters of the MXG book |  |
| ADOC\* | Documentation for the various types of records processed - ADOC30 for example documents all of the datasets and variables created from type 30 SMF records |  |
| ANAL\* | These members perform some sort of analysis and reporting. Some are straight SAS code but some are SAS macros with parameters that let you tailor the analysis | Yes |
| ASM\* | Assembler programs  |  |
| ASUM\* | Summarization routines - ASUM42DS for example summarizes the TYPE42DS dataset to 15 minute intervals an exception to this is ASUMUOW which combines CICS DB2 and MQ records into a single record for each unit of work that can then be summarized with ASUMCICX. | Yes |
| AUTO\* | SAS Autoexec members for different SAS releases and operating systems |  |
| BLD\* BUILD\* | Members that BUILD something. BUILDPDB is the original program that builds the performance data base |  |
| CHANGExx | Change members for each version |  |
| CHANGES | Changes to the most current annual version  |  |
| CHANGESS | All changes back to the dawn of time |  |
| CONFI\* | SAS Config members for different SAS releases and operating systems |  |
| DAY\* DAILY\* | Various daily kinds of routines - infrequently used |  |
| DIFF\* | In many cases data is accumulated across intervals and must be 'diffed' to determine what happened in any given interval. These members perform that function. |  |
| DOC\* | Documentation for various things. Some are presentations made at SHARE/CMG and some just document products. |  |
| EO\* | Output members for VM/XA datasets | YES |
| ERBRMF\* | Sample RMF parmlib members |  |
| EX\* | For every dataset created by MXG when reading raw data there is an EX\*\*\*\*\*\* member that defines the conditions under which the dataset is output. Usually it will have something like: OUTPUT \_TY30U1; but you could modify it to output only batch work by specifying IF TYPETASK='JOB' THEN OUTPUT \_WTY30U1; | YES |
| FORMATS | This is a critical member of the SOURCLIB. If contains the code to build the formats used by MXG to turn HEX strings in the SMF data into something user intelligible, to calculate MIPS and other things.  |  |
| FTP\* | Examples of FTP jobs to send data to MXG Support | YES |
| GDG\* | Examples of JCL to run MXG using GDG datasets for output | YES |
| GRAF\* | Members that produce graphs of various data in various formats - some are MACROs with parameters that will let you use SAS/GRAPH or ODS GRAPHICS. Others are just examples of code. | YES |
| IHDR\* | Some SMF and other records have headers that can be used to limit the data being read or correct errors in the data. | YES |
| IMAC\* | Every type of data has an IMAC member. This is the place where you can tell us what SMF record type some user record might be or make other changes. Some of these you may need to touch in the tailoring of MXG to your site but most need no attention. | YES |
| JCL\* | Sample JCL members for zOS. It is not expected that these will work in your environment without some changes. | YES |
| MNTH\* MONTH\* | Summarization routines - summarizations to the month level. |  |
| MOB\* | NEW - members to help you get discounts from IBM for mobile work -  | YES |
| MXG\* | PROCs and pieces of code to run MXG. MXGNAMES can be used with the SAS provided SAS PROCs and there are some other ways to avoid having a separate MXG PROC.  | YES |
| QA\* | Code used in the QA process for MXG - it is not expected that you will ever use them but you could |  |
| RMFINTRV | Builds the RMFINTRV dataset - you will need to modify this member during the tailoring of MXG | YES |
| SEND\* | How to send problem reports and data to support@mxg.com | YES |
| SMFDUMP | A sample exit for the dumping of SMF MAN datasets. USE AT YOUR OWN RISK. | YES |
| TESS\* TEST\* | Test code used by the QA\* members |  |
| TRND\* | Long term trend building - highly summarized and condensed data - the original spec was 5 years data in 50 cylinders. It has grown since then but is still small. | YES |
| TYPE\* | The code to read a 'type' of record. TYPE30 reads the type 30 SMF records. Resulting data Is (generall) in the WORK LIBNAME. |  |
| TYPS\* | Same as the TYPE\* but output will (generally) be in the PDB LIBNAME. |  |
| U\* UTIL\* | Utility programs.  |  YES |
| VGET\* | A SAS macro that 'gets' some piece of information. VGETOBS for example returns the number of observations in a SAS dataset. |  |
| VMAC\* | The code used by the TYPE TYPS and BUILD membes to read a 'type' of record. VMAC30 describes the datasets and variables in the type 30 30 SMF records amd contains the code to read them.. |  |
| VMXG\* | A SAS macro that performs some function. VMXGSUM for example does summarization. |  |
| X\*  | Extra members that may have been submitted by a user |  |
| Z\* | Old members generally no longer used |  |
| ZZZZZZZZ | Z last member in Z SOURCLIB. It it is missing something failed in your install. |   |

NOTE that some members have a YES under user modifiable. These are members you may need or want to change when it comes to tailoring MXG. Other members should not be changed unless directed to do so by support@mxg.com.

Now you know the pattern of names so what are you looking at? It is all source code (including the assembler modules and you may need one or two of those depending on your installation.) For the most part, MXG is written using substitution style SAS macros. A substitution macro is a very simple construct. A name that equates to a piece of code.

For example:

MACRO \_ETY30U1 %%INCLUDE SOURCLIB(EXTY30); % <- contained in the VMAC30 member of the SOURCLIB

In the program will resolve to ‘OUTPUT \_WTY30U1;’ along with some comments. The double % tells the compiler that it is really a single % sign and the trailing % sign terminates the substitution macro.

Most MXG programs are little more than a chain of substitution macros put together to create a program. Using the type 30 SMF data as an example the member TYPE30 is:

 %INCLUDE SOURCLIB(VMACSMF,VMAC30,IMACKEEP);

DATA

\_VAR30 <- comes from VMAC30

\_SMF <- comes from VMACSMF

\_CDE30 <- comes from VMAC30

;RUN;

The \_xxxxxxx macros are all contained in VMACSMF or VMAC30 and IMACKEEP can be used to modify the underlying substitution macros. Each type of SMF (or other data sources) has a VMAC member in the SOURCLIB that contains an \_VAR macro describing the datasets and variables and an \_CDE macro that is the code to read the record. There are many exits to allow you to modify the code and they will be described in some detail later.

But, say you want to suppress the output of the type30\_4 dataset (step termination data.) In the original release of MXG you would have inserted in IMACKEEP:

 MACRO \_ETY30U4 %

So that there would be no output statement for the type30\_4 dataset. Today however, there is no need to edit IMACKEEP. You can much more simply use:

%LET MACKEEP=%QUOTE(

 MACRO \_ETY30U4 %

);

 %INCLUDE SOURCLIB(TYPE30);

To accomplish the same result. The indentation is my personal choice for using MACKEEP and I always wrap it in a %QUOTE to avoid issues with special characters. It could be on a single line so long as there is a space between the % and the ). The IMACKEEP member now has at the end of the member

 &MACKEEP

Which resolves the SAS MACRO variable MACKEEP into MACRO \_ETY30U4 %.

That brings us to SAS MACROs (real macros that can contain logic and both local and global macro variables.) MXG now has global macro variables defined in the VMXGINIT member used when you start up an MXG session that define the default output locations for datasets (generally PDB) and many other bits of information needed for the processing of data. Most can be modified by you using a simple

%LET name=new value;

%LET is the equivalent of A=B in the base language and may be used in open code. MXG makes extensive use of the macro facility building ‘shortcuts’ for common coding problems and giving you simpler ways of tailoring MXG. The important ones will each have their own chapter.

But, before you begin using this source code, MXG must first be installed. That has always been a simple process – it is the tailoring to your site that becomes complex. MXG will run on any platform where SAS will run. In fact, it has been demonstrated many times that it may actually run faster on a Windows PC reading the SMF data directly from a tape on zOS using FTP than the same job would run on the zOS system. So, since many people are migrating SAS from zOS to Windows or LINUX we will address both options.

# Chapter 3: Installing MXG – zOS or ASCII?

The choice is yours. SAS can be perceived as very expensive on zOS – particularly if the only use is to run MXG. You can run as easily on a Windows or UNIX PC or server (it does not have to be a server license of SAS only a workstation.) In fact, in many cases MXG will run faster on a dedicated PC than on a mainframe running zOS. The choice is likely going to come down to money. MXG itself is probably the most inexpensive piece of mainframe centric software you can find but whether you choose to run with SAS or WPS that software can be very expensive. Another factor to consider is data storage. zOS does a very good job of data management while ASCII platforms are a bit less robust. But, if you are connected to a SAN or have large disk drive, MXG provides some data management tools that make the task of managing the data simpler. So let the dollars drive the decision.

## Chapter 3a: Installing MXG on zOS

First things first – you have to get the file containing MXG. Send an email to admin@mxg.com for instructions on downloading the most current version of MXG. You will want either EBCxxyy.zip or the tersed version of the PDS TERxxyy.TER where xxyy is the current version number such as 32.05.

An historical footnote. The first few years it happened coincidentally that when it was time for the 2nd annual version there had been an interim release 2.1 so the annual release became 2.2. Likewise year 3 it just happened there had been 2 interim releases so it became 3.3. And the pattern has been preserved though there will not be 31 interim releases the next annual release will be 32.32. Interim releases tend to center on some major new product release from IBM or other vendors or when we make a major mistake and have to correct an error condition (mea culpa.)

### Method 1 – IEBUPDTE

If you downloaded the EBCxxyy.zip or EBCxxyy.ebc (10 times larger than the zip) then your JCL to build the SOURCLIB would look like:

|  |
| --- |
|  //STEP1 EXEC PGM=IEBUPDTE,PARM=NEW **//SYSPRINT DD DUMMY <IF POINTED TO SYSOUT, PLAN FOR 900,000+ LINES** **//\* AS IT WILL PRINT THE ENTIRE MXG SOURCE LIBRARY!** //SYSIN DD DSN=EBCxxyy.EBC,DISP=SHR //SYSUT2 DD DSN=MXG.V3205.MXG.SOURCLIB, // DISP=(NEW,CATLG), // RECFM=FB,LRECL=80,BLKSIZE=0, // SPACE=(CYL,(350,25,1199)) |

### Method 2 – TERSE UNPACK

|  |
| --- |
| //UNTERSE EXEC PGM=TRSMAIN,PARM='UNPACK'//SYSPRINT DD SYSOUT=\*//INFILE DD DSN=MXG.TER3205.TERSED,DISP=SHR//OUTFILE DD DSN=MXG.V3205.MXG.SOURCLIB,UNIT=SYSDA,// DISP=(NEW,CATLG),RECFM=FB,LRECL=80,BLKSIZE=0,// AVGREC=M,SPACE=(80,(3,1,1199)) PDS: 3 MIL 80 BYTE RECS//\* 32.05: 335 CYL |

Either method has the same result. Which you use is a matter of choice. The only remaining step is to build the MXG FORMAT library. That can be done with the following JCL

### Running PROC FORMAT on zOS

|  |
| --- |
| /FORMATS EXEC SAS,ENTRY=SAS,// CONFIG='MXG.V3205.MXG.SOURCLIB(CONFIGV9)'//SASLOG DD SYSOUT=\*//SASLIST DD SYSOUT=\*//SOURCLIB DD DSN=MXG.V3205.USERID.SOURCLIB,DISP=SHR// DD DSN=MXG.V3205.MXG.SOURCLIB,DISP=SHR//LIBRARY DD DSN=MXG.V3205.MXG.FORMATS,// UNIT=SYSDA,DISP=(NEW,CATLG),SPACE=(CYL,(12,2))//\* NORMALLY YOU SHOULD LET SAS SET THE DCB ATTRIBUTES FOR YOU, BUT//\* IF YOU HAVE PROBLEMS WITH SMS RULES, THIS IS A CORRECT DCB://\* DCB=(RECFM=FS,LRECL=27648,BLKSIZE=27648),//SYSIN DD \* %INCLUDE SOURCLIB(FORMATS); |

The dataset names used are only suggestions. You clearly have to conform to the dataset naming standards in your shop so modify these as you need. But, you should NEVER use a DCB attribute on a SAS dataset unless your SMS rules get in the way.

## Chapter 3b: Installing MXG on an ASCII platform

Installing MXG on ASCII is in some ways simpler than on zOS and in others a touch more complex but it is still a very simple process.

Again first things first – you have to get the file containing MXG. Send an email to admin@mxg.com for instructions on downloading the most current version of MXG. You will want DIRxxyy.zip where xxyy is the current version number such as 32.05.

Once you have the ZIP file you will need to decide where to install MXG. You will need several directories. The recommended structure is:

 C:\MXG

 \SOURCLIB

 \USERID

 \FORMATS

What disk drive you choose to use is completely arbitrary. It can be a network drive or a local drive it really does not make a lot of difference. Unzip the downloaded file into C:\MXG\SOURCLIB and MXG is almost installed.

Next you will need to build an AUTOEXEC. Start with the AUTOEXEC member in the SOURCLIB.

The statements that are required are shown here in **RED** and recommended options are in **GREEN**. The rest are used in the original BUILDPDB process which we are going to avoid and can safely be omitted.

|  |
| --- |
| /\* CONCATENATE USERID.SOURCLIB DIRECTORY AHEAD OF MXG.SOURCLIB \*/ /\* LAST UPDATED: DEC 7, 2007. CHANGE 25.267. \*/ /\* \*/ /\* AUTOEXEC.SAS FOR ASCII EXECUTION OF MXG IS LIKE \*/ /\* YOUR JCL FOR Z/OS EXECUTION, SINCE YOU NAME "DDS" HERE,\*/ /\* CONFIGV9 FOR Z/OS EXECUTION, SINCE THE OPTIONS YOU SET \*/ /\* THERE ARE SET IN THIS FILE. \*/ /\* YOU CAN PUT THIS FILE IN YOUR ROOT DIRECTORY FOR SAS, OR \*/ /\* USE SAS -AUTOEXEC C:\MXG\SOURCLIB\AUTOEXEC.SAS TO INVOKE. \*/ /\* \*/ /\* REQUIRED FILENAME/LIBNAME FOR MXG SOURCE, FORMAT, LOCATIONS \*/**FILENAME SOURCLIB ('C:\MXG\USERID' 'C:\MXG\SOURCLIB');****LIBNAME LIBRARY 'C:\MXG\FORMATS';** /\* FOLLOWING EXAMPLES ARE FOR MVS AND BUILDPDB PROCESSING OF SMF DATA\*/ /\* THAT WAS DOWNLOADED OR USING FTP ACCESS METHOD TO READ DIRECT: \*/ /\* 1. TO READ DOWNLOADED Z/OS SMF DATA FROM ASCII DISK FILE AND TO \*/ /\* ALWAYS HAVE AN INFILE SMF POINTING TO AN SMF FILE FOR TESTING: \*/ FILENAME SMF 'C:\MXG\SMFDATA\SMFSMALL.U' RECFM=S370VBS LRECL=32760 BLKSIZE=32760; /\* 2. TO READ CONCATENATED DOWNLOADED SMF FILES, USE THIS SYNTAX FILENAME SMF ( 'C:\MXG\SMFDATA\SMFSMALL.U' 'C:\MXG\SMFDATA\SMFSMAL2.U' ) RECFM=S370VBS LRECL=32760 BLKSIZE=32760; \*/ /\* 3. TO USE FTP ACCESS METHOD TO READ SMF DATA WITHOUT DOWNLOAD: FILENAME SMF FTP ("'SYS1.SMF'" "'SYS2.SMF'" ... ) USER='XXXXXX' HOST='YYYYYYY' DEBUG S370VS RCMD='SITE RDW' LRECL=32760 PASS='XXXXXXXX'; NOTE: IF YOUR SMF DATA IS ON TAPE, YOU SHOULD USE: RCMD='SITE RDW READTAPEFORMAT=S' \*/LIBNAME PDB 'C:\MXG\PDB';LIBNAME CICSTRAN 'C:\MXG\CICSTRAN';LIBNAME SPIN 'C:\MXG\SPIN';LIBNAME DB2ACCT 'C:\MXG\DB2ACCT '; /\* MXG REQUIRED FOR SOME PROGRAMS - CREATE AS ZERO LENGTH \*/FILENAME INSTREAM 'C:\MXG\USERID\INSTREAM.SAS';/\* MXG REQUIRED FOR MONTHBLD \*/LIBNAME DUMMY 'C:\MXG\DUMMY '; /\* FOLLOWING EXAMPLES ARE FOR VM/ESA (AND VM/XA) PROCESSING \*/FILENAME MWINPUT 'C:\MXG\VMDATA\MONWRITE.U' RECFM=F LRECL=4096 BLKSIZE=28672; **/\* RECOMMENDED OPTIONS, CAN BE CHANGED FOR PERFORMANCE \*/****OPTIONS STIMER; /\* CAN BE NOSTIMER. PRINTS ELAPSED TIME ON LOG \*/****OPTIONS FULLSTIMER; /\* CAN BE NOFULLSTIMER. PRINTS CPU/MEMORY ON LOG\*/****OPTIONS SORTSEQ=ASCII; /\*JUST FOR SAFETY\*/****OPTIONS SORTEQUALS; /\* SEE CHANGE 25.028. \*/****OPTIONS COMPRESS=YES; /\*ASCII PLATFORMS FASTER ELAPSED, LOW CPU COST\*/****OPTIONS SORTSIZE=400M; /\*(80% OF INSTALLED MEMORY FOR PERFORMANCE)\*/****OPTIONS VALIDVARNAME=V7; /\*VARIABLE NAMES TO BE UP TO 32 CHARS\*/** **/\* OPTION MIMIZES THE DIAGNOSTICS PRINTED ON SAS LOG. \*/** **/\* REMOVE THREE "NO" FOR FULL DIAGNOSICS TO SEND TO SUPPORT@MXG.COM\*/****OPTIONS SOURCE NOSOURCE2 NOMPRINT NOMLOGIC;** **/\* OPTION IN SAS V9+ TO PRINT CURRENT TIME ON LOG/LIST \*/****OPTIONS DTRESET;** **/\*BELOW ARE MXG-REQUIRED OR MXG-VERY STRONGLY RECOMMENDED\*/****OPTIONS DLDMGACTION=REPAIR;****OPTIONS YEARCUTOFF=1960;****OPTIONS DKROCOND=NOWARN;****OPTIONS ERRORS=2;** **/\*OPTIONS MAUTOSOURCE SASAUTOS=(SOURCLIB '!SASROOT\CORE\SASMACRO');\*/****OPTIONS MAUTOSOURCE SASAUTOS=(SOURCLIB SASAUTOS);****%INCLUDE SOURCLIB(VMXGINIT); %VMXGINIT;****RUN;** |

Once the autoexec is constructed you can make a copy of the SAS shortcut on your desktop and change the properties to be:

"C:\Program Files\SASHome\SASFoundation\9.3\sas.exe" -CONFIG "C:\Program Files\SASHome\SASFoundation\9.3\nls\en\sasv9.cfg" -autoexec 'd:\mxg\userid\autoexec.sas'

The addition of the –AUTOEXEC pointing at your AUTOEXEC (mine is on the D drive) means that when SAS starts, it will automatically start with everything you need to run MXG. If you then click on the ICON you should see something like this:

|  |
| --- |
| NOTE: Copyright (c) 2002-2010 by SAS Institute Inc., Cary, NC, USA.NOTE: SAS (r) Proprietary Software 9.3 (TS1M2) Licensed to MERRILL CONSULTANTS, Site 70006668.NOTE: This session is executing on the W32\_7PRO platform.NOTE: Enhanced analytical products:SAS/STAT 12.1NOTE: SAS initialization used: real time 1.27 seconds cpu time 0.84 secondsNOTE: AUTOEXEC processing beginning; file is d:\mxg\userid\autoprod.sas.NOTE: Libref LIBRARY was successfully assigned as follows: Engine: V9 Physical Name: D:\MXG\FORMATSNOTE: Libref PDB was successfully assigned as follows: Engine: V9 Physical Name: D:\MXG\PDBNOTE: Libref CICSTRAN was successfully assigned as follows: Engine: V9 Physical Name: D:\MXG\CICSTRANNOTE: Libref SPIN was successfully assigned as follows: Engine: V9 Physical Name: D:\MXG\SPINNOTE: Libref DB2ACCT was successfully assigned as follows: Engine: V9 Physical Name: D:\MXG\DB2ACCTNOTE: Library DUMMY does not exist.MXGNOTE: VMXGINIT LAST UPDATED: JUL 29, 2014. CHANGE 32.178.NOTE: The macro VMXGINIT completed compilation without errors. 46168 instructions 923452 bytes.ODS=AUTO AND SYSIN=WELCOME TO MXG SOFTWARE, FROM MERRILL CONSULTANTS, DALLAS, TEXASTECH SUPPORT: 214 351 1966 SUPPORT@MXG.COM WWW.MXG.COMMXG 32.07 DATED AUG 3, 2014 HAS BEEN SUCCESSFULLY INITIALIZED.NOTE: Fileref= SOURCLIB Physical Name= D:\CHANGES Physical Name= D:\V3207NOTE: Fileref= SASAUTOS Physical Name= C:\Program Files\SASHome\SASFoundation\9.3\core\sasmacro Physical Name= C:\Program Files\SASHome\SASFoundation\9.3\aacomp\sasmacro Physical Name= C:\Program Files\SASHome\SASFoundation\9.3\accelmva\sasmacro Physical Name= C:\Program Files\SASHome\SASFoundation\9.3\dmscore\sasmacro Physical Name= C:\Program Files\SASHome\SASFoundation\9.3\graph\sasmacro Physical Name= C:\Program Files\SASHome\SASFoundation\9.3\stat\sasmacroNOTE: The macro TIMEBILD completed compilation without errors. 84 instructions 4656 bytes.NOTE: The macro VMXGTIME completed compilation without errors. 13 instructions 348 bytes.NOTE: The macro VMXGVERS completed compilation without errors. 25 instructions 692 bytes.NOTE: AUTOEXEC processing completed. |

Now that you have started MXG the first thing to do is an include of the FORMATS member to build the FORMATS catalogue used by MXG. Simply put %INCLUDE SOURCLIB(FORMATS); in the program window and press PF3.

You may also find it convenient to change the ICON to one of Barry’s smiling face:



MXG is now installed and ready for you to begin customization.

# Chapter 4: Customizing MXG

Customization of MXG is largely done in IMAC\*\*\*\* members. There is an IMAC for each type of data and some other more general IMACs and those more general ones are where customization typically takes place. The important ones are IMACSHFT (shift definitions), IMACKEEP (many different things can go here), and IMACWORK (obsolete workload definitions now replaced by RMFINTRV).

## IMACSHFT

Determines how you want shifts defined and how long those shifts are. 40 years ago we needed to know what was going on at the shift level since there were people involved working the various shifts mounting tapes, loading paper into printers, changing print ribbons, responding to messages and so on. In today’s world of lights out data centers, auto operators, auto tape handling, it is much less of an issue and may or may not be important to your company. The stock IMACSHFT member defines shifts as

|  |
| --- |
|  P 9HR ( PRIME) - 08:00 THRU 16:59 MON-FRI  N 15HR (NONPRIME) - 17:00 THRU 07:59 MON-FRI (TIL SAT AM) W 48HR (WEEKEND ) - 08:00 SAT THRU 07:59 MON  H 24HR (HOLIDAY ) - IF DATE IS IN THE HOLIDAY TABLE  |

The logic inside the IMACSHFT member can be modified to suit your particular company.

|  |
| --- |
| LENGTH SHIFT $1 SHFTTIME 8; FORMAT SHFTTIME DATETIME21.2; IF DATETIME=. THEN DATETIME=.; SHFTTIME=DATETIME; DROP SHFTTIME MXGDURTM; /\*--------------------------------------------------------\*/ /\* PRIME SHIFT: WEEKDAYS 08:00 TO 16:59 \*/ /\* MONDAY-FRIDAY (DAY 2-6) \*/ /\* THE SAS WEEKDAY FUNCTION RETURNS 1 FOR SUN, 7 FOR SAT \*/ /\*--------------------------------------------------------\*/ IF ( 8 LE HOUR(TIMEPART(DATETIME)) LE 16 AND 2 LE WEEKDAY(DATEPART(DATETIME)) LE 6 ) THEN DO; SHIFT='P'; DATETIME=DHMS(DATEPART(DATETIME),8,0,0); MXGDURTM=9\*3600; END; /\*--------------------------------------------------------\*/ /\* NONPRIME SHIFT: WEEKDAYS 17:00 TO 23:59 \*/ /\* MONDAY-FRIDAY (DAY 2-6) \*/ /\* OR 00:00 TO 07:59 ON \*/ /\* TUESDAY-SATURDAY (DAY 3-7) \*/ /\*--------------------------------------------------------\*/ ELSE IF ( ( HOUR(TIMEPART(DATETIME)) GE 17 AND 2 LE WEEKDAY(DATEPART(DATETIME)) LE 6 ) OR ( 0 LE HOUR(TIMEPART(DATETIME)) LE 7 AND 3 LE WEEKDAY(DATEPART(DATETIME)) LE 7 ) ) THEN DO; SHIFT='N'; IF HOUR(TIMEPART(DATETIME)) LT 17 THEN DATETIME = DHMS(DATEPART(DATETIME)-1,17,0,0); ELSE DATETIME = DHMS(DATEPART(DATETIME),17,0,0); MXGDURTM=15\*3600; END; /\*--------------------------------------------------------\*/ /\* WEEKEND SHIFT: ALL TIMES NOT DEFINED ABOVE \*/ /\*--------------------------------------------------------\*/ ELSE DO; SHIFT='W'; IF WEEKDAY(DATEPART(DATETIME)) EQ 7 THEN DATETIME = DHMS(DATEPART(DATETIME),8,0,0); IF WEEKDAY(DATEPART(DATETIME)) EQ 1 THEN DATETIME = DHMS(DATEPART(DATETIME)-1,8,0,0); ELSE IF WEEKDAY(DATEPART(DATETIME)) EQ 2 THEN DATETIME = DHMS(DATEPART(DATETIME)-2,8,0,0); MXGDURTM=(24+24)\*3600; END; LABEL SHIFT='SHIFT\*OF\*START'; /\*--------------------------------------------------------\*/ /\* HOLIDAY SHIFT: ALL HOURS ON DATES IN TABLE, IF THE \*/ /\* COMMENT BLOCK IS REMOVED TO ENABLE. \*/ /\*--------------------------------------------------------\*/ /\* TABLE OF DATES FOR HOLIDAYS. YOU MUST UPDATE THE LIST EACH YEAR\*/ /\* SEE MEMBER NEWUSER FOR ADDITIONAL HOLIDAY CONSIDERATIONS. \*/ /\* NOTE THAT SHFTTIME IS USED, WHICH HAS THE ORIGINAL DATETIME. \*/ /\* THE SHIFT LOGIC CAN RESET DATETIME TO THE START-OF-SHIFT VALUE. \*/ &MACSHFT; |

Say you wanted to change to have 2 12 hour shifts per day Mon-Fri and a weekend shift. You could change the logic to:

|  |
| --- |
| /\*--------------------------------------------------------\*/ /\* PRIME SHIFT: WEEKDAYS 08:00 TO 19:59 \*/ /\* MONDAY-FRIDAY (DAY 2-6) \*/ /\* THE SAS WEEKDAY FUNCTION RETURNS 1 FOR SUN, 7 FOR SAT \*/ /\*--------------------------------------------------------\*/ IF ( 8 LE HOUR(TIMEPART(DATETIME)) LE 19 AND 2 LE WEEKDAY(DATEPART(DATETIME)) LE 6 ) THEN DO; SHIFT='P'; DATETIME=DHMS(DATEPART(DATETIME),8,0,0); MXGDURTM=43200; /\* 12 hours \*/ END; /\*--------------------------------------------------------\*/ /\* NONPRIME SHIFT: WEEKDAYS 20:00 TO 23:59 \*/ /\* MONDAY-FRIDAY (DAY 2-6) \*/ /\* OR 00:00 TO 07:59 ON \*/ /\* TUESDAY-SATURDAY (DAY 3-7) \*/ /\*--------------------------------------------------------\*/ ELSE IF ( ( HOUR(TIMEPART(DATETIME)) GE 19 AND 2 LE WEEKDAY(DATEPART(DATETIME)) LE 6 ) OR ( 0 LE HOUR(TIMEPART(DATETIME)) LE 7 AND 3 LE WEEKDAY(DATEPART(DATETIME)) LE 7 ) ) THEN DO; SHIFT='N'; IF HOUR(TIMEPART(DATETIME)) LT 17 THEN DATETIME = DHMS(DATEPART(DATETIME)-1,17,0,0); ELSE DATETIME = DHMS(DATEPART(DATETIME),17,0,0); MXGDURTM=43200; END; /\*--------------------------------------------------------\*/ /\* WEEKEND SHIFT: ALL TIMES NOT DEFINED ABOVE \*/ /\*--------------------------------------------------------\*/ ELSE DO; SHIFT='W'; IF WEEKDAY(DATEPART(DATETIME)) EQ 7 THEN DATETIME = DHMS(DATEPART(DATETIME),8,0,0); IF WEEKDAY(DATEPART(DATETIME)) EQ 1 THEN DATETIME = DHMS(DATEPART(DATETIME)-1,8,0,0); ELSE IF WEEKDAY(DATEPART(DATETIME)) EQ 2 THEN DATETIME = DHMS(DATEPART(DATETIME)-2,8,0,0); MXGDURTM=(24+24)\*3600; END; LABEL SHIFT='SHIFT\*OF\*START'; |

And then if you wanted to add a table of holidays you could add:

|  |
| --- |
|  IF DATEPART(SHFTTIME) IN ( '01JAN2014'D '26MAY2014'D '04JUL2014'D '01SEP2014'D '27NOV2014'D '28NOV2014'D '25DEC2014'D '26DEC2014'D '01JAN2015'D ) THEN DO; SHIFT='H'; MXGDURTM=86400; END; |

You may also find it useful to build a user format for reporting that better describes you shift definitions. I cannot count the number of times I have been asked ‘what does SHIFT=P mean?’ You can avoid this question by building a format in the IMACFMT member as shown here for the 12 hour and holiday shifts:

|  |
| --- |
| PROC FORMAT LIB=LIBRARY;VALUE $SHIFT ‘P’=’Mon-Fri 8AM-5PM’ ‘N’=’Mon-Fri 5PM-8AM’ ‘W=’Sat-Sun’ ‘H’=’Holiday’; |

Then when you run the FORMATS member IMACFMT will be included and $MYSHIFT format will be placed in the FORMATS catalog. Adding FORMAT SHIFT $MYSHIFT.; to the IMACSHFT member will ensure that SHIFT is always printed using the formatted values. It does not take extra space in the datasets - that is still a single byte. Alternatively, you can build your own FORMAT library and use the FMTSEARCH option to include your own FORMAT library;

 OPTIONS FMTSEARCH=(WORK LIBRARY MYFORMTS);

Which should be place in the IMACINIT member of your USERID.SOURCLIB. I have always found it simpler to use the LIBRARY LIBNAME to hold all the formats but that is a choice. Be careful if you do this to avoid conflicts with MXG formats all of which start with either MG or $MG.

If SHIFT values are not important to you, simply make the IMACSHFT member:

 SHIFT=’ ‘;

 LABEL SHIFT=’ ‘;

There are many MXG members and reports that are expecting the variable SHIFT to exist but the values in the variable are unimportant other than in reports.

## IMACKEEP

The most common use of IMACKEEP is as a place to store the IDs of user SMF records. IBM records have a record type between 0 and 128 while user records range from 129-255. By default since there is no standardization of these records all default to 512 – an impossible value for this field. In the SOURCLIB for each of the records you will find an IMAC\*\*\*\* member which contains code looking like:

/\* MACRO \_IDACF2 512 % \*/

This is from the IMACACF2 member and you could make the change in the IMAC member but, it may be simpler to put all of the MACRO \_ID\*\*\* code in IMACKEEP so that you have a single repository of all of the user SMF records.

Some other uses of IMACKEEP can include modifying the KEEP lists for datasets. Say you wanted to drop the 3350 DASD EXCP and IO time variables from the TYPE30\_4 dataset.

 MACRO \_KTY30U4 DROP=EXCP3350 IOTM3350 %

Is all you need to insert in IMACKEEP. So why this example? MXG keeps old things around forever. It has been a long time since it was possible to even define a 2305 or 3330 or 3350… device to the operating system but the variables tracking them will still exist albeit with zeroes or missing values. Since the default in MXG is to compress SAS datasets those fields will not use a great deal of space but you may wish to (carefully) prune back the variables that are kept in some datasets.

RMFINTRV/IMACWORK

IMACWORK was the original way to define workloads for building the RMFINTRV dataset – a 1 observation per interval summary of all of the important RMF data. It was limited to 15 workloads:

 BAT

 CICS

 IMS

 TSO

 OTH0-OTH9

 OTHR

15 workloads seems like a lot but many sites wanted more granularity as well as more flexibility in naming the variables so the VMXGRMFI macro was created which allows up to 999 additional workloads to be defined in addition to the 15 in IMACWORK (using IMACWORK is optional.) While the use of performance groups is still available (I have actually seen current data from a system that old) it is more usual to use service classes to define workloads. The parameters for VMXGRMFI may seem obscure but when this was written there was still an 8 byte limit on parameter names.

 -

|  |
| --- |
| INTERVAL=QTRHOUR OR THESE OTHER TIME DURATION INTERVALS: ANNUAL, SEMIANN, QUARTER, MONTH, WEEK, DATE, SHIFT, EIGHTHR, FOURHOUR, THREEHR, TWOHOUR, HOUR, HALFHOUR, TWENTYMIN, QTRHOUR , TENMIN, FIVEMIN, MINUTE, SECOND.BUT THE INTERVAL= VALUE MUST BE EQUAL TO OR LARGERTHAN THE DURATM OF YOUR RMF DATA. YOU CANNOT USEINTERVAL=QTRHOUR IF YOUR DATA IS WRITTEN HOURLY,NOR CAN YOU USE INTERVAL=QTRHOUR WITH 10 MIN DATA. AND IF YOU HAVE MULTIPLE VALUES OF RMF INTERVALS,THE INTERVAL= ARGUMENT MUST BE AN INTEGER WHEN ITIS DIVIDED BY EACH ACTUAL RMF INTERVAL. SO IF YOU HAVE BOTH 10 MINUTE AND 15 MINUTE RMF DATA, YOU MUST USE INTERVAL=HALFHOUR (OR INTERVAL=HOUR).This is by far the most common usage of the INTERVAL-Parameter but there are some other options.INTERVAL=DETAIL INTERVAL=DETAIL CREATES ONE OBS IN PDB.RMFINTRV FOR EACH RAW RMF INTERVAL, I.E. EACH RAW STARTIME SO THERE IS NO SUMMARIZATION BY TIME, WHICH ISTHE OLD INTERVAL=DURSET WITH UNCHANGED IMACRMFI.THE DURATION SPECIFIED WILL ALSO BE PLACED IN THE LABEL OF THE OUTPUT DATASET.INTERVAL= ONLY AFFECTS PDB.RMFINTRV SUMMARY.SYNC59=YES IF YOU HAVE EVER USED MICS OR YOU HAVE VERY LARGE SMF VOLUMES SUCH THAT POPPING THE CICS & THE RMF INTERVALS ON THE HOUR CAN CAUSE SMF CONGESTION, YOU MAY HAVE CHOSEN TO SYNCH SMF/RMF INTERVALS TO 59. THIS IS ESPECIALLY COMMON IN MICS SHOPS. IF YOU USE THE INTERVAL CAPABILITIES TO SET THE RMFINTRV DATA TO QUARTER OR HALF HOUR THIS CAN RESULT IN DATA BEING IN THE WRONG INTERVAL. FOREXAMPLE, 13:59 USING A QTRHOUR INTERVAL WOULD BE THE INTERVAL STARTING AT 13:45. IF THIS IS NOT WHAT YOU WANT OR EXPECT, SPECIFY SYNC59=YES, AND THE CREATION OF PDB.RMFINTRV WILL ADD 60 SECONDS TO THE STARTIME BEFORE THE INTERVAL IS CREATED, SO IT WILL BE RESET TO THE 00/15/30/45, ETC., AND SMF70GIE WILL ALSO BE RESET TO 00/15/30/45. |

Now you have defined the intervals you want to use for the RMFINTRV dataset and it is time for the ‘hard part’. Defining the workloads. Workloads are defined with several different parameters.

The first is IMACWORK. If it is YES, IMACWORK workload definitions are used. That does not mean you are limited to 15 workloads since you can still use the up to 999 WORKxxx= parameters to define more workloads. But, the more you use the more cumbersome the data will become and possibly the less useful it will be. IMACWORK defaults to a value of NO.

NOTYPE74 also defaults to NO. In data centers of the past there were limits to the number of devices that could be connected to the operating system and getting the average response time for all the DASD devices was a useful measure. That is data from the type 74 RMF records. Today with 10s of thousands of DASD and other devices, that average value tends to be less useful and it can be a mountain of data. But, if you want DASD data in RMFINTRV, set NOTYPE74 to YES.

The next question is how to define the workloads. There are 3 possibilities with Workload Manager.

1. Use control service classes
2. Use report service classes
3. Use both

Option 3 should be avoided since it invariably leads to double counting resources somewhere in the data where there is an overlap between control and report classes. It is very strongly recommended that you use report classes. Why? It provides you with much better granularity in the data and they are essentially no cost with no possibility of double dipping. The caveat is that for this to be possible each and every classification rule must have some report class specified – if not explicitly then a default value for the subsystem should be provided. There are two parameters that control this:

 USEREPRT – the DEFAULT value is NO

 USECNTRL – the DEAFULT value is YES

Choose the option appropriate to your site and then we can start defining workloads with the WORKxxx= parameter. There can be up to 999 WORK parameters. Each has several pieces separated by a /. The basic format is:

 WORK1=varname/label/perf group/service class/# periods/system/sysplex/workload

So, if I specify:

 WORK1=PRODBAT/PROD BATCH//BATHI BATWRM BATMED/3

I have told VMXGRMFI that this workload should have variable names that begin with PRODBAT (PRODBATCPU PRODBATIOTM….), the variables should be labelled ‘PROD BATCH\*CPU\*TIME’ for PRODBATCPU, there are no performance groups (and hopefully you are current enough to not ever need them), the service classes BATHI BATWRK and BATMED all will be placed in this workload, and there are 3 periods in at least one of these service classes.

Adding system/sysplex/workload names is optional but if they are there the service class is only used if it also matches those specifications. So:

WORK1=PRODBAT/PROD BATCH//BATHI BATWRM BATMED/3/SYSA SYSB

WORK2=PRODBAT/PROD BATCH//BATHOT PRODBAT//SYSC

Says that for SYSA and SYSB PRODBAT is these service classes on SYSA and SYSB and on SYSC is the BATHOT and PRODBAT service classes with only a single period. (NOTE: this can only happen if the 3 systems are not in the same SYSPLEX.)

An example of RMFINTRV:

Assume a SYSPLEX where RMF is set to 15 minute intervals synched to 59 with the following report classes:

|  |
| --- |
|  SYSTEM – things we cant control SYSSTC - things we can control that we really want to run at high priority CICSPROD - velocity goal service class for PROD CICS TRANA - response time goal service class for some PROD CICS Transactions TRANB - response time goal service class for other PROD CUCS  TRANHOT - response time goal service class for the most loved PROD CICSCICSTEST - velocity goal service class for test CICS controlled at the REGION level (very important) DB2P - PROD DB2 DB2T - Test DB2  TSO - TSO Users – 3 periods DDFP - Prod DDF – 2 periods DDFT - Test DDF – 3 periods BATHOT - Prod batch  BATWRM - Prod batch but less important BATTEST - Test Batch |

My RMFINTRV member would look like:

|  |
| --- |
| %VMXGRMFI( INTERVAL=QTRHOUR, SYNCH59=YES, USECNTRL=NO, USEREPRT=YES, WORK1=PRODCICS/Prod CICS//CICSPROD TRANA TRANB TRANC TRANHOT, WORK2=TESTCICS/Test CICS//CICSTEST, WORK3=SYSTEM/System Tasks//SYS:, WORK4=PRODDB2/Prod DB2//DB2P, WORK5=TESTDB2/Test Db2//DB2T, WORK6=PRODBAT/Prod Batch//BATHOT BTWRM, WORK7=TSO/TSO//TSO/3. WORK8=DDFPROD/DDF Prod//DDFP/2, WORK9=DDFTEST/DDF Test//DDFT//3, WORK10=TESTBAT/Test Batch/BATTEST ); |

Notice that in the WORK3 definition the service class is specified as SYS:. The ‘:’ is the SAS equivalent of a wild card so that workload will pick up any report class starting with SYS. Parameters are separated by commas and the sub-parameters with a /. The last parameter must NOT have a comma and the macro call is terminated with a ‘);’.

There are many other possible parameters to use with VMXGRMGI and all documented within the SOURCLIB member but for the beginner, these are all that is really required. When we discuss trending we will bring this example back and enhance it to build a long term trend.

So why is it important that test CICS service classes be managed at the region level rather than the transaction level (which happens to be the default in WLM)? Early on with WLM we found that the overhead of running WLM was very high. The primary cause was CICS. WLM builds a PCB (performance control block) for every possible value of MAXTASK for every CICS region. So, if you have lots of test CICS regions with very high MAXTASK values the PCB chain can get very long and WLM overhead can become a major workload on your system. Setting the service class to REGION suppresses that behavior.

## UTILWORK

If you happen to also be the person that designed your Workload Manager policy, setting up the RMFINTRV member in your USERID.SOURCLIB will be relatively simple. If not, you will need to gain an understanding of the WLM policy at your site to correctly configure RMFINTRV. The UTILWORK member in the MXG SOURCLIB can help by building a skeleton of the RMFINTRV member that you can then edit. To execute UTILWORK and build the skeleton you can either read the TYPE72 RMF records from SMF or the TYPE72GO dataset from an existing PDB.

|  |
| --- |
| //JOBCARD JOB WHATEVER WORKS FOR YOU  //STEP1 EXEC MXGSASV9  //SMF DD DSN=YOUR SMF DATA  OR  //PDB DD DSN=YOUR PDB  //RMFINTRV DD DSN=WHERE YOU WANT THE SKELETON STASHED  //SYSIN DD \*  %UTILWORK(PDB=SMF);  OR  %UTILWORK(PDB=PDB);  |

By default, UTILWORK looks at service classes but you can tell it to use report classes by adding USERPRT=YES to the macro invocation.

The resulting RMFINTRV member might look like this:

|  |
| --- |
| %VMXGRMFI( INTERVAL=QTRHOUR, USEREPRT=NO, USECNTRL=GOAL, WORK1=WORK1/BOTTOM//BOTTOM/1 , WORK2=WORK2/CICSFAST//CICSFAST/1 , WORK3=WORK3/CICSPROD//CICSPROD/1 , WORK4=WORK4/DDFBATCH//DDFBATCH/3 , WORK5=WORK5/DDFHI//DDFHI/3 , WORK6=WORK6/DDFLOW//DDFLOW/2 , WORK7=WORK7/OMVS//OMVS/1 , WORK8=WORK8/REPLICAT//REPLICAT/1 , WORK9=WORK9/REPLTEST//REPLTEST/1 , WORK10=WORK10/SERVERS//SERVERS/1 , WORK11=WORK11/STCHIGH//STCHIGH/1 , WORK12=WORK12/STCLOW//STCLOW/1 , WORK13=WORK13/STCMED//STCMED/1 , WORK14=WORK14/SYSSTC//SYSSTC/1 , WORK15=WORK15/SYSTEM//SYSTEM/1 , WORK16=WORK16/TMHI//TMHI/1 , WORK17=WORK17/TMLOW//TMLOW/2 , WORK18=WORK18/TMMED//TMMED/1 , WORK19=WORK19/TMWRM//TMWRM/2 , WORK20=WORK20/TSO//TSO/2 , WORK21=WORK21/TSOHOT//TSOHOT/2 , WORK22=WORK22/UNKNOWN//UNKNOWN/1); |

UTILWORK has no way of knowing what variable names you want to use so simply calls it the same as the WORKx parameter. Likewise, it can have no knowledge of what labels should be applied so uses the name of the service class. There is also no combining of service classes into workloads. For example, in this case, I know that TMHI TMMED and TMWRM (Thruput Manager service classes) are all production batch. Clearly some editing is required but UTILWORK will give you working syntax as a starting point.

In this instance, I want to combine CICSFAST and CICSPROD into a single workload for CICS. CICSFAST is the response time service class which will have transaction counts and response times and CICSPROD is the velocity service class that will have all of the CPU time. I also want to combine all of the SYS\* and STC\* service classes, all of the TSO\* service classes, and UNKNOWN (things that fall through the classification rules) and BOTTOM.

The edited RMFINTRV member would look like this:

|  |
| --- |
|  %VMXGRMFI( INTERVAL=QTRHOUR, USEREPRT=NO, WORK1=CICSP/PROD CICS//CICSFAST CICSPROD, WORK2=DDFP/PROD DDF//DDFBATCH DDFHI, WORK3=DDFT/TEST DDF//DDFLOW/2, WORK4=REPLP/PROD REPL//REPLICAT/1, WORK5=REPLT/TEST REPL//REPLTEST/1, WORK6=SERVR/SERVERS//SERVERS/1, WORK7=SYST/SYSTEM TASKS//STC: SYS: OMVS, WORK8=PRODBAT/PROD BATCH//TMHI TMMED TMWRM/1, WORK9=TESTBAT/TEST BATCH//TMLOW/2, WORK10=TSO/TSO//TSO:/2, WORK11=UNKN/UNKNOWN//UNKNOWN BOTTOM/1); |

## ANALACTM

ANALACTM can also be useful in gaining an understanding of your WLM policy. It will produce a report showing you the relative importance and goals of the various service classes.



ANALACTM can be invoked as %ANALACTM; to run against an existing PDB or as %ANALACTM(PDB=SMF) to read your SMF data and create this report. As with many other MXG macros, there are many more parameters that are documented in the member. If you happen to have SAS/GRAPH, ANALACTM will also produce some graphic representations of performance based on a presentation by Rich Olcott at CMG some years ago. See change 26.064 in the CHANGESS member. If SAS/GRAPH is not present, the graphs are suppressed but if you do have SAS/GRAPH and do not want the graphs simply add CHART=NO to the parameters.

With the tailoring of RMFINTRV, most of the required tailoring of MXG to get started is now complete. But before we can begin adding extra features and user records it is time to talk assembler modules.

# Chapter 5: Assembler Modules

There are several assembler modules included In the MXG SOURCLIB but there are only 4 you are likely to need. ASMTAPEE, ASMRMFV, EXITCICS, and EXITMON6. Each serves a different purpose and may or may not be useful to you.

## ASMTAPEE – the MXG Tape Mount Monitor

How long does it take to mount tapes? How many tape drives are concurrently allocated? What jobs are seeing the most delay from tape mounts? All questions that can be answered with the tape mount monitor – but in these days of virtual tape and automated tape libraries this may or may not be of interest to you. The SMF data generated by the mount monitor is automatically included as part of the BUILDPDB process but can be easily suppressed (more on that later.)

In addition to tracking tape mounts, the monitor can also capture SYSLOG messages for mounts and such other events as you may choose. A recent enhancement adds the ability to track page dataset usage by tasks and MSU consumption by importance. Have an assembler guru review the member if your assembler skills are as rusty as mine.

 There are lots of comments to guide you but the important thing to change is:

//ASM EXEC PGM=ASMA90,

// PARM='DECK,NOOBJ,XREF(SHORT),NOUSING,SYSPARM(238,ES6)'

238 is the SMF record type the monitor will write unless you change it to some other unused number. You will likely need to change some of the dataset names including the load library for the LKED step and you will need to add a job card but other than that it should be just a matter of submitting the job. The job will create the MXGTMNT PROC and the load module In the load library of your choice. Starting the monitor (assuming you put the PROC in a PROCLIB that is in the JES concatenation) is just a matter of S MXGTMNT.

IT IS IN NO WAY REQUIRED THAT YOU RUN THIS MONITOR! It is a matter of choice. The resources consumed are minimal – 3-4 CPU seconds an hour on a large system.

## ASMRMFV – RMF III datasets

RMF III data can be very useful in solving problems and MXG car read that data but first, the VSAM data stores for RMF III must be read and decompressed into a flat file for MXG to process. The JCLRMFV member in the SOURCLIB has a substantial body of documentation on the care and feeding to MXG of RMF III datasets.

## EXITCICS – CICS/DB2 Decompression exit

Both CICS and DB2 can have compressed SMF data and it is a very very good idea to compress this data. DB2 accounting data is often 80-90% of the total volume of data in many shops and compressing it can save large amounts of space and dumps of SMF data. If LOGGER is in use for recording SMF it may be less of an issue but could still be very valuable. Problem though is it has to be decompressed in order to read it. SAS has routines that will handle the decompression but the CPU cost on zOS can be very steep. EXITCICS provides a SAS INFILE exit (CICSFIUE) that will do the decompression for you at much less cost on zOS. The decompression costs on ASCII are much lower (and you cannot run a zOS INFILE exit on ASCII in any case.)

To make use of this exit you must modify the dataset names in the JCL and provide a load library for the module which must then also be part of the STEPLIB concatenation when you execute SAS/MXG. If you see a message on your SASLOG saying:

ERROR: YOUR DB2 RECORDS ARE COMPRESSED…

Then the EXIT is not in use and MXG will consume considerably more CPU than is necessary to read your DB2 data. If on the other hand you see:

 MXGNOTE: SMF EXIT CICS IS IN USE.

MXG found the INFILE exit in the STEPLIB and will use that exit to decompress the data.

## EXITMON6 – Landmark Monitor Decompression

Just like CICS and DB2 the Landmark TMON products compress the data stores and in order to read them you need this INFILE exit in place. It is essentially the same process as EXITCICS so there is no further discussion here but there is (as always) documentation in the SOURCLIB member.

.

# Chapter 6: Tailoring BUILDPDB in 1987 – the hard way

Let’s say you wanted to add the type 1415 42 and HSM data to your daily PDB. In the original version of MXG you had to copy these members to your USERID.SOURCLIB and modify them.

 EXPDBINC

 EXPDBVAR

 EXPDBCDE

 EXPDBOUT

## EXPDBINC would have looked like:

 %INCLUDE SOURCLIB(VMAC1415,VMAC42,VMACHSM);

## EXPDBVAR would be:

MACRO \_VARUSER

 \_VAR1415 \_VAR42 \_VARHSM

%

## EXPDBCDE would be:

 MACRO \_CDEUSER

 \_CDE1415 \_CDE42 \_CDEHSM

 %

## And finally EXPDBOUT:

 MACRP \_EPDBOUT

\_S1415 \_S42 \_SHSM

 %

If you are fluent in SAS, this is probably simple but, if you are not what does it all mean? In each case, it is simply adding the building blocks contained in the VMAC\*\*\*\* members to the existing BUILDPDB code.

With later versions of MXG it became possible to specify simply:

## Using MACKEEP to override EXPDB members

|  |
| --- |
| %LET EPDBINC=%QUOTE( %INCLUDE SOURCLIB(VMAC1415,VMAC42,VMACHSM););%LET EPDBVAR=%QUOTE( MACRO \_VARUSER \_VAR1415 \_VAR42 \_VARHSM %);%LET EPDBCDE=%QUOTE( MACRO \_CDEUSER \_CDE1415 \_CDE42 \_CDEHSM %);%let EPDBOUT=%quote( MACRO \_EPDBOUT \_S1415 \_S42 \_SHSM);%INCLUDE SOURCLIB(BUILDPDB); |

);

Hmmmm, doesn’t seem much simpler. Surely there is a better way?

# Chapter 7: Using UTILBLDP

There is now a better/simpler way! UTILBLDP which is now the recommended way to tailor BUILDPDB.

To accomplish the same coding – adding 1415, 42, and HSM records to BUILDPDB all that is needed is:

%UTILBLDP(USERADD=1415 42 HSM/251,OUTFILE=INSTREAM);

%INCLUDE INSTREAM;

And you are done! So let’s examine the usage of UTILBLDP and the parameters available.

## UTILBLDP Parameters

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Default Values** | **Usage** |
| BUILDPDB | YES | YES – invokes BUILDPDBNO - does NOT run BUILDPDBJES3 – invokes BUILDPD3 |
| USERADD |  | SMF records you want to add. For IBM records (ID=0 to 128) all you need is the xxxx portion of the VMACxxxx member. For example, to add the 14 and 15 records you would code USERADD=1415. For record types greater than 128 you can specify both the xxxx portion of the VMAC member and the record type. For example, if you wanted to add HSM and the first of the two HSM records is 251 you would code USERADD=HSM/251 – the / separates the mnemonic for the record and the record ID. When specifying multiple records the records are separated by spaces as in: USERADD=1415 HSM/251 |
| SPINCNT | 0 | The SPIN count to use for jobs – replaces IMACSPIN. If you are not running BUILDPDB it is ignored. |
| SPINUOW | 0 | The SPIN count for ASUMUOW. If you are not running CICS transactions and ASUMUOW it is ignored. |
| TIMEDIF |  | Normally not needed but if there are differences in the clocks on systems it may be needed to correctly match the type 30 and type 26 records for jobs. See IMACTIME for more discussion. |
| SUPPRESS |  | Suppresses record types you don’t want BUILDPDB to process. Sometimes size dictates that CICS and DB2 be processed separately from the daily BUILDPDB. SpecifyingSUPPRESS=110 DB2 will do that. |
| WANTSMF |  | List of SMF recored IDs and subtypes to be read. This allows you to choose which datasets have observations in them when all is done. It is best used when you are only trying to read a few subtypes. USERADD=42 70 73 74,  WANTSMF=42.6 70.1 73 74.1 74.5 74.7 ,This list becomes the MACFILE exit. |
| ZEROOBS |  | Causes listed SMF types to have 0 OBS in the output datasets by suppressing the \_E\*\*\*\*\*\* exit. For example:ZEROOBS=74 all type 74 datasets would have 0 OBSZEROOBS=74.1 74.5 – TYPE74 and TYPE74CA datasets would have 0 OBS |
| ECHO | NO | YES – list the generated code on the SASLOGNO – do NOT list the generated code on the SASLOG |
| RMFINTRV | YES | If BUILDPDB=YES run RMFINTRVIF USERADD= includes the RMF data types it will also include RMFINTRV |
| IMACEXCL |  | If your CICS data is tailored – the flat files containing the tailored code to read that data. |
| OUTFILE | BUILDMXG | The DDNAME/FILENAME where you want the generated source code to be stored. Could be SOURCLIB(MYSOURCE), a DDNAME or FILENAME, but the most common usage is INSTREAM. |
| MACFILEX |  | Anything you want to put in MACFILE can be put here or you can put it in the IMACFILE member or in a %LET MACFILE= statement. |
| MACKEEPX |  | Anything you want to put in MACKEEP goes here or you can put it in IMACKEEP or use a %LET MACKEEP= statement. |
| MACDB2HX |  | Anything that could be inserted in the IMACDB2H member goes here |
| MAC110HX |  | Anything belonging in the IMAC110H member goes here |
| IMACKEEP |  | If you want to keep the generated MACKEEP logic for this program this is where you specify where to put it. Something like SOURCLIB(MYKEEP). |
| EXPDBOUT |  | Code that gets inserted ahead of the EPDBOUT macro and before the \_S macros are invoked (rarely used) |
| SORTOUT | YES | YES – sort the output datasets using the \_S macros into the destination PDBNO – skip the sorts and leave the data in WORK |
| MXGINCL | ASUMUOW ASUMCICX ASUM70PR ASUMCACH ASUMTAPE ASUMTMNT ASUMTALO ASUMDBAA ASUMDBSS | Members that will be automatically included after reading the data if the correct type of data is being processed. If TYPE74 is suppressed (for example), ASUMCACH is not run. |
| INCLAFTR |  | SOURCLIB members YOU wish to include after running the code. |

## Some examples of using UTILBLDP.

**Example 1:**

Run BUILDPDB but suppress the type CICS and DB2 data, create type 74 datasets with 0 OBS, while including the type 42 data.

|  |
| --- |
| %UTILBLDP(USERADD=42,SUPPRESS=110 DB2,ZEROOBS=74,OUTFILE=INSTREAM);%INCLUDE INSTREAM; |

**Example 2:**

Read the 1415 21 and TMNT data then run ASUMTAPE.

|  |
| --- |
| %UTILBLDP(BUILDPDB=NO,USERADD=1415 21 TMNT/238,INCLAFTR=ASUMTAPE, OUTFILE=INSTREAM); %INCLUDE INSTREAM; |

**Example 3:**

An incredibly complex example but shows the things you can do with UTILBLDP. Use the CICS INFILE exit to read CICS and DB2 data but suppress the detail CICSTRAN and DB2 Accounting datasets while gathering the statistics data. Add to BUILDPDB the 1415 (keeping only selected variables, 42 (keeping only the TYPE42DS dataset), HSM data, 6156 data, ENTIREX data with ID 160, SYNCSORT data with type 208, and the subtypes of 102 data necessary for the DB2 Audit reports in ANALDB2R. Suppress the automatically included members in MXGINCL but include ASUM70PR and ASUMTAPE after running BUILDPDB.

|  |
| --- |
| %LET SMFEXIT=CICS; /\* User the CICS INFILE Exit \*/%LET MACKEEP=%QUOTE( MACRO \_BTY1415 JOB DSNAME % /\* change the type1415 sort \*/ MACRO \_VTY1415 /\* Change the KEEP list for TYPE1415 \*/ KEEP=JOB DSNAME LRECL BLKSIZE DSORG RECFM trksaloc temp opentime smftime trksaloc blkcnt keep % \_N42 /\* suppress all TYPE42 dataset \*/ MACRO \_WTY42DS PDB.TYPE42DS % /\* put type42DS directly in PDB without a SORT \*/ MACRO \_LTY42DS PDB.TYPE42DS % MACRO \_S42 % /\* Suppress type 42 SORTS \*/ MACRO \_KTY6156 GATLIMIT GATCNT GDGATTR % /\* Add to TYPE6156 keep list \*/ MACRO \_ETY30MU % /\* Suppress the TYPE30MU dataset \*/ MACRO \_WCICTRN \_NULL\_ % /\* suppress CICSTRAN and CICSBAD \*/ MACRO \_LCICTRN \_NULL\_ % MACRO \_WCICBAD \_NULL\_ % MACRO \_LCICBAD \_NULL\_ % MACRO \_SCICBAD % MACRO \_WDB2ACC \_NULL\_ % /\* suppress all DB2 accounting datasets \*/ MACRO \_LDB2ACC \_NULL\_ % MACRO \_SDB2ACC % MACRO \_WDB2ACP \_NULL\_ % MACRO \_LDB2ACP \_NULL\_ % MACRO \_SDB2ACP % MACRO \_WDB2ACB \_NULL\_ % MACRO \_LDB2ACB \_NULL\_ % MACRO \_SDB2ACB % MACRO \_WDB2ACG \_NULL\_ % MACRO \_LDB2ACG \_NULL\_ % MACRO \_SDB2ACG % MACRO \_WDB2ACR \_NULL\_ % MACRO \_LDB2ACR \_NULL\_ % MACRO \_SDB2ACR % MACRO \_WDB2ACW \_NULL\_ % MACRO \_LDB2ACW \_NULL\_ % MACRO \_SDB2ACW % MACRO \_LCICLDR PDB.CICLDR % /\* keep the CICSLDR dataset detail in the PDB \*/ MACRO \_ECICLDR IF LDRFC GT 0 THEN OUTPUT \_WCICLDR; % /\* if the count of loads is GT 0 \*/ ); /\* End MACKEEP \*/ %LET ADD30U4=SMF30PFR SMF30PRJ; /\* add variables to the STEPS/JOBS summarization \*/ %LET ADD30U5=SMF30PFR SMF30PRJ; %UTILBLDP( USERADD=16 HSM 6156 TCP SYNC/208 ENTX/160 1415 42 102.23 102.24 102.25 102.55 102.83 102.87 102.105 102.107 102.140 102.141 102.142 102.143 102.144 102.145 102.169 102.269 102.270 102.319, OUTFILE=INSTREAM, MXGINCL=,INCLAFTR=ASUM70PR ASUMTAPE); %INCLUDE INSTREAM; |

# Chapter 8: BLDSMPDB

When you use the examples in chapter 7 to create the code to run BUILDPDB it writes that code out to INSTREAM and in that code you would see a %include sourclib(BUILDPDB) to invoke the BUILDPDB process. You can also make use of BLDSMPDB which on ASCII allows you to do everything in a single ‘job’. BLDSMPDB can be used on zOS to run DAILY, WEEKLY, and MONTHLY processes and to limit the dataset retained at various levels. Parameters in **GREEN** are ASCII only and are ignored on zOS.

## BLDSMPDB Parameters

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Default Value** | **Usage** |
| **SMFIN** |  | **Only used on ASCII systems and points to the location of your SMF data – if not specified the SMF filename is used which is the recommended method** |
| **FIRSTRUN** | **NO** | **Used on ASCII systems when AUTOALOC is being used** |
| **AUTOALOC** | **NO** | **ASCII only will use VMXGALOC to allocate pseudo-GDG datasets if you specify YES and manage them based on the criteria you supply** |
| **BASEDIR** | **C:\MXG\** | **Where will the PDB directories be allocated** |
| **BASECICS** | **C:\MXG\** | **Where will the CICS directory be allocated (if blank reverts to the BASEDIR value)** |
| **BASEDB2** | **C:\MXG\** | **Where will the DB2 directory be allocated (if blank reverts to the BASEDIR value)** |
| **DATEFMT** | **DATE7.** | **VMXGALOC builds directories with names like Dddmmmyy for the daily PDB. You may want to make this YYMMDD so that the directories will be listed in chronological order.** |
| **DAY2KEEP** | **14** | **How many daily PDBs are retained** |
| **CICSKEEP** | **14** | **How many daily CICSTRAN datasets are retained** |
| **DB2KEEP** | **14** | **How many daily DB2ACCT datasets are retained** |
| **WEK2KEEP** | **12** | **How many WEEKLY PDBs are retained** |
| **MTH2KEEP** | **100** | **How many MONTHLY PDBs are retained** |
| RUNDAY | YES | Run the daily process – if NO daily process is not run – if PDB then reads SMF data into PDB but that is all |
| RUNWEEK | NO | NO – do not run WEEEKLY processYES - Combine last 7 days into WEEKWTD – Add today’s data to week – if first day of week start process over  |
| RUNMNTH | NO | NO – do not run MONTHLY processYES – combine last 5 weeks and last 7 days into MONTHMTD – add today’s data to month – if second day of month start process overFORCE – run MONTHLY process on a day other than 1st of MONTH |
| RUNTRND | WEEKLY | When should the TREND database be update? Change to DAILY to process each day or blank to suppress TRENDing. |
| FORCEDAY |   | The date in SAS date format which you want MXG to think is the day you are running – 01SEP14 for example to rerun Sept 1 daily/weekly/monthly |
| RERUN |  | If you are using AUTOALOC=NO and you are doing a rerun on a day other than the day the data should have been processed then you have to tell us where to put the data in the static PDB directories – MON TUE WED THU FRI SAT SUN  |
| SORTEDBY | NO | If YES the SORT order in the PDB is preserved in the WEEKLY/MONTHLY processing. The SORT order for most datasets is not terribly useful in reporting – it is primarily to remove duplicates – and does increase the complexity of WEEK/MONTH processing. |
| DCOLLECT |  | If you want to add DCOLLECT data to the PDB specify DCOLLECT=DCOLLECT and the data will be read from the DCOLLECT or specify DCOLLECT = fully qualified DSNAME and it will by dynamically allocated |
| TMC |  | It you want to add CA1 data to the PDB specify TMC=TMC and the data will be read from the TMC FILENAME or specify TMC=fully qualified DSNAME and it will be dynamically allocated |
| RMM |  | It you want to add RMM data to the PDB specify RMM=RMM and the data will be read from the RMM FILENAME or specify RMM=fully qualified DSNAME and it will be dynamically allocated |
| CONTROLT |  | It you want to add CONTROLT data to the PDB specify CONTROLT=CONTROLT and the data will be read from the CONTROLT FILENAME or specify CONTROLT=fully qualified DSNAME and it will be dynamically allocated |
| DAILYDSN |  | If you added DCOLLECT and tape catalog data to the PDB do you want to also run DAILYDSN? YES or TMC looks for TMC data, RMM for RMM data, CONTROLT for CONTROLT. |
| DALYKEEP |  | At the end of the normal BUILDPDB process there is a PROC COPY to copy the data to the day of the week which it represents when you are running with AUTOALOC=NO. So if you are running on Monday it copies the data to the SUN LIBNAME. You can limit the datasets that are kept by specifying them here. For example: DALYKEEP=JOBS STEPS CICS: would keep only the JOBS STEPS and all datasets starting with CICS |
| ROLLWEEK | YES | If you are running with static LIBNAMES when it comes time to run the weekly, WEEK4 is copied to WEEK5, WEEK3 to WEEK4, WEEK2 to WEEK3, and WEEK1 to WEEK2. WEEK1 then becomes the new week. This is only applicable if AUTOALOC=NO. |
| WEEKSTRT | &STARTDAY | The day of the week on which your week starts. Traditionally for MXG that is MON and &STARTDAY will resolve to MON unless you have changed the default. If you want your week to start on some other day you can specify SUN MON TUE WED THU FRI or SAT |
| WEEKTAPEWTDTAPE | NO | zOS only – use TAPETEMP logic to build weekly datasets. TAPETEMP logic avoids multiple tape mounts by building the datasets in sequential format on disk and appending them to a tape dataset. |
| WEEKKEEP | \_ALL\_ | List the SAS datasets that you want to keep at a weekly level |
| WEEKDROP | SPIN: SPUN: | List the datasets you do NOT want to keep at the weekly level. WEEKKEEP and WEEKDROP can both be used. Use whichever results in the shorter list of datasets. A : is a wildcard in SAS. |
| WEEKNSRT |  | A list of datasets where the SORT order should be ignored. SORTEDBY=NO has this effect on all datasets |
| WEEKCODE |  | A stub of code to be executed after WEEKLY processing |
| MNTHTAPEMTDTAPE | NO | zOS only – use TAPETEMP logic to build monthly datasets. TAPETEMP logic avoids multiple tape mounts by building the datasets in sequential format on disk and appending them to a tape dataset. |
| MNTHKEEP | \_ALL\_ | List the SAS datasets that you want to keep at a monthly level |
| MNTHDROP | SPIN: SPUN: | List the datasets you do NOT want to keep at the monthly level. MNTHKEEP and MNTHDROP can both be used. Use whichever results in the shorter list of datasets. A : is a wildcard in SAS. |
| MNTHNSRT |  | A list of datasets where the SORT order should be ignored. SORTEDBY=NO has this effect on all datasets |
| MNTHCODE |  | A stub of code to be executed after MONTHLY processing |
| MONTH1 |  | Where it last month? Only required if you are using static LIBNAME/DDNAMEs |
| TRNDCODE |  | Stub of code to run after TRENDing |
| RUNDAYS |  | MXG always assumes you will process the data every day but if you don’t want to run every day specify the days you want it to run as MON TUE WED THU FRI SAT to exclude Sunday. |
| BUILDPDB | BUILDPDB | If BUILDPDB then the normal BUILDPDB is included and executed But you can change that to run tailored code built by UTILBLDP and output to INSTREAM by specifying BUILDPDB=INSTREAMBUILDPDB=COPYONLY skips the processing of SMF data and copies the contents of the PDB to the appropriate day of the week (static allocations only).BUILDPDB=NONSMF tells us you only want to process something like DCOLLECT or TMC – non-SMF data sources. |
| AUTOINCL | ASUM70PR ASUMUOW ASUMCICX ASUMDB2A ASUMDB2B ASUMDBSS ASUMJOBS ASUMCACH ASUMTAPE ASUMTMNT ASUMTALO | Members that are automatically included after BUILDPDB – if you don’t want or need any of these they can be excluded. When you use UTILBLDP this list will be altered. |
| AUTOTRND | TRNDCEC TRNDCELP TRNDCICX TRNDDB2A TRNDDB2P TRNDDBSS TRNDRMFI TRND72GO TRNDTMNT TRNDTALO | Members that are automatically included for TRENDing – if you don’t want or need any of these they can be excluded. |
| BASEPDB | PDB | When running WEEKLY/MONTHLY processing the program looks here to see what datasets exist and extracts their SORTBY information in order to build the code for WEEKLY/MONTHLY processing. |
| ERASEPDB | YES | Clean out the existing PDB LIBNAME before processing when RUNDAY=YES |
| **PDB****MON****TUE****WED****THUFRI****SAT** **SUN****WEEK1****WEEK2****WEEK3****WEEK4****WEEK5****CICSTRAN****DB2ACCT** | **C:\MXG\PDB****C:\MXG\MON****C:\MXG\TUE****C:\MXG\WED****C:\MXG\THU****C:\MXG\FRI****C:\MXG\SAT****C:\MXG\SUN****C:\MXG\WEEK1****C:\MXG\WEEK2****C:\MXG\WEEK3****C:\MXG\WEEK4****C:\MXG\WEEK5****C:\MXG\CICSTRAM** | **ONLY used if AUTOALOC=NO and only on ASCII platforms. The places you want to put the output data.****DB2ACCT defaults to PDB** |
| **REROUTE1-REROUTE8** |  | **Allows you to reroute datasets to different locations. For each dataset there is a dddddd value unique to that dataset. There are three portions of this parameter: dddddd!directory!nosort. The third is optional – in the case of a large dataset like DB2ACCTP you may not want to sort. For example if you want to route DB2ACCTP to your H drive without sorting you would specify:****REROUTE1=DB2ACP!H:\!NOSORT** |

## BLDSMPDB Examples

Example 1: Using Example 1 from UTILBLDP to run a daily job for the first time using AUTOALOC=YES

|  |
| --- |
| %UTILBLDP(USERADD=42,SUPPRESS=110 DB2,ZEROOBS=74,OUTFILE=INSTREAM);%BLDSMPDB(BUILDPDB=INSTREAM, AUTOALOC=YES,BASEDIR=H:\MXG,FIRSTRUN=YES); |

Example 1A: Using Example 1 from UTILBLDP to run a daily job for after the first time using AUTOALOC=YES

|  |
| --- |
| %UTILBLDP(USERADD=42,SUPPRESS=110 DB2,ZEROOBS=74,OUTFILE=INSTREAM);%BLDSMPDB(BUILDPDB=INSTREAM, AUTOALOC=YES,BASEDIR=H:\MXG); |

FIRSTRUN=YES sort of primes the pump and builds all of the directories that are expected for weekly/monthly processing and fills them with 0 OBS copies of all of the datasets in the PDB.

Example 2: Add weekly/monthly processing with the week starting on Sunday

|  |
| --- |
| %UTILBLDP(USERADD=42,SUPPRESS=110 DB2,ZEROOBS=74,OUTFILE=INSTREAM)%BLDSMPDB(BUILDPDB=INSTREAM, AUTOALOC=YES,BASEDIR=H:\MXG,RUNWEEK=YES,RUNMNTH=YES,WEEKSTRT=SUN); |

Chapter 9: AUTOALOC\VMXGALOC

On a zOS platform the ideal (and in many cases required) method of allocating and managing PDB datasets is the use of generation data groups (GDGs). That allows the system to manage the retention and storage of datasets based on SMS rules and the catalog definitions of the GDGs. Unfortunately that structure does not exist on ASCII platforms (at least not yet.) The management of space on disk falls onto the user. MXG solves that problem with VMXGALOC.

VMXGALOC creates directories and then manages them based on the criteria you give it. Directories are created and deleted over time in the place and at the times you specify.

## VMXGALOC Parameters

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Default Value** | **Usage** |
| DATEFMT | DATE | The format of the date in the directories that are built. The daily directory for example would be D05SEP14. You may want to change this to a value that puts the directories in a more rational sequence. YYMMDD perhaps makes more sense. |
| FIRSTRUN | YES | Always forced to YES. In the first iteration of this macro it was necessary for the first run. Now it looks to see if the directory it is expecting is there and if not creates it. |
| FORCEDAY |  | The day of the run you want to make if other than today. Can be a SAS date value like 05SEP14 or a relative number such as TODAY()-2. |
| BASEDIR | C:\MXG | Where do you want the data to reside? All of the other directories are built here |
| BASECICS | C:\MXG | Where do you want the data to reside? The CICS directory is built here |
| BASEDB2 | C:\MXG | Where do you want the data to reside? The DB2 directory is built here |
| WEEKSTRT | &STARTDAY | The day of the week on which your week starts. &STARTDAY will resolve to MON unless you have changed it. |
| WEEKKEEP | 12 | How many weeks of data to keep |
| DAYSKEEP | 14 | How many days of data to keep |
| MNTHKEEP | 15 | How many months of data to keep |
| CICSKEEP | 14 | How many days of CICSTRAN to keep |
| DB2KEEP | 14 | How many days of DB2ACCT to keep |
| TRENDING |  | When do you do trending?Blank – no trendingDAILY – as part of daily cycleWEEKLY – as part of weekly cycleMONTHLY – as part of monthly cycleTREND directories will inherit the lifetime of the cycle you specify |
| RUNWTD | NO | Yes/NO run week to date processing |
| RUNMTD | NO | YES/NO run month to date processing |
| CLEARALL | YES | CLEAR all allocated libnames |
| READONLY | NO | If YES then no aging of directories is performed |

VMXGALOC is only useful on an ASCII platform and simulates the behavior of a GDG on a zOS system. As shown in the BLDSMPDB examples it can be used in conjunction with BLDSMPDB or it can be used by itself for allocating LIBNAMEs for reporting.

## VMXGALOC Examples

Example 1: Assume you have suppressed the reading of DB2 data by BUILDPDB but want to read it and place it in a different directory and you only want to keep 5 days of data all of it in the same directory.

|  |
| --- |
| %VMXGALOC(BASEDIR=C:\DB2,dayskeep=5,weekkeep=,db2keep=,cicskeep=,mnthkeep=, TRENDING=); %READDB2(PDBOUT=PDB); /\* READDB2 will be discussed later \*/ |

Example 2: Assume you want to run reports against data from the daily PDB..

|  |
| --- |
| %VMXGALOC(BASEDIR=C:\DB2,readonly=yes); Your report code |

# Chapter 10: How much space will I need?

As with most things the answer is ‘it depends’. In most shops today, the vast majority of data comes from either or both CICS and DB2. In many instances, DB2 accounting data can account for 80-90% of all of the SMF data written on a system.

The DB2 accounting records are large and even compressed can take a huge amount of space. For example, at one relatively small site on a recent day there were 23,132,039 SMF records written in total. Of that number, 16,995,444 were DB2 accounting records and 148,180 were CICS records (but one CICS record is many transactions while one DB2 record is one transaction.) At this site on that day 15GB of data was written to SMF and the CICS and DB2 data are both compressed and the DB2 accounting data was 79.3% of the total bytes written to SMF.

Normally, on zOS, the DB2 accounting datasets and the CICS transaction data would be written to tape so for now we can exclude them from the calculation. For this site which uses the final UTILBLDP example in the previous chapter, at the time the SMF data is consolidated into daily datasets a dataset specific to BUILDPDB is created that excludes the DB2 and CICS transaction data and other record types that are not used in the BUILDPDB job. That left a total of 4,284,213 records to be read for a total of 3.7GB. The resulting PDB dataset was 2233 cylinders or about 1.8GB with SAS compression enabled. So in this instance the space required was about 50% of the size of the SMF data that was needed to create the PDB. Your mileage may vary depending on the types of data you add to the BUILDPDB process and how well the data compresses. You could use the ANALID member of the MXG SOURLCIB to make an estimate of the size you would need based on the record types you intend to process (ANALID will be discussed later.) A good starting point seems to be the size of the data to be read times .5 (1/2) excluding CICS and DB2 transaction data.

But what about weekly and monthly data? The answer again – ‘it depends’. Now it depends largely on how much of the detail data you want or need to keep at these levels.

In the author’s view there are several kinds of data in the PDB.

|  |  |  |
| --- | --- | --- |
| Type of data | Usage | Lifetime |
| Detail data  | Primarily problem solving | 2 weeks |
| Accounting data – detail | Chargeback | As many years as auditing requires |
| Summary/trend data | Capacity planning | As many years as may be needed for a planning horizon |

Does all of this data need to be rolled up into a week or a month? Probably not in most cases. The WEEKBLD and MONTHBLD processes in MXG if untailored copy everything that is expected to be in the base PDB but will not copy any datasets you may have added without tailoring of the members. If you user BLDSMPDB to control the weekly and monthly processes the default is to copy all of the datasets but you have the ability to pick and choose which dataset to keep at those levels and which to exclude using the WEEKKEEP/WEEKDROP and MNTHKEEP/MNTHDROP parameters. Why two? So that you can choose the shorter path. If you only want to exclude a few dataset the WEEEKDROP would be a shorter list but it you want to keep only a few then WEEKKEEP would be the shorter list.

# Chapter 11: Running BUILDPDB on zOS

One of the strengths of zOS is data management. SMS manages datasets migrating them from primary DASD to level 1 (compressed DASD) and ultimately level 2 (tape), keeps the catalogs current, and generally manages the DASD space on the system. In the original MXG structure the assumption was there would be fixed dataset names such as:

 MXG.PDB

 MXG.MON-MXG.SUN – 1 for each day of the week

 MXG.WEEK MXG.WEEK1-MXG.WEEK5 – weekly datasets

 MXG.MONTH – monthly dataset

 MXG.TREND – a trend dataset

But that puts the burden on the user to manage the space, make backups, make copies of the month data for each month, etc. On zOS it makes the most sense (and is required by standards in many if not most shops) to create the PDB datasets using GDGs (generation data groups.) Then just as you can choose how long to keep the various pieces of the PDBs on ASCII with VMXGALOC you can do the same thing on zOS with GDGs. So how do you create a GDG?

## JCL to create a GDG

You will also find this JCL in the MXG SOURCLIB member JCLSPGDG and depending on the options you choose you may not need all of these

|  |
| --- |
| //JCLSPGDG JOB 'WHATEVER WORKS'//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*///\* ZOS ONLY - NO ASCII EQUIVALENT - SEE CHANGE 29.105. \*///\* JCLSPGDG - RUN ONCE TO CREATE GDGS, AND THEN NEVER AGAIN, UNLESS \*///\* THERE IS A NEED TO ALTER A GDG BASE OR TO CHANGE THE \*///\* Z/OS GDG DATASET NAME. \*///\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*LTIPLE P\*\*\*\*\*\*\*\*\*\*\*\*\*///S1 EXEC PGM=IDCAMS//SYSPRINT DD SYSOUT=\*//\* DEFINES THE GDGS FOR EXECUTION ON ZOS//\* MXG.DAILY.PDB - DAILY PDB//\* MXG.DAILY.SPIN - DAILY SPIN//\* MXG.DAILY.IOPDB - DAILY IO PDB//\* MXG.DAILY.DB2ACCT - DAILY DB2ACCT//\* MXG.DAILY.CICSTRAN - DAILY CICSTRAN//\* MXG.DAILY.CICS - DAILY CICS PDB//\* MXG.DAILY.MQPDB - DAILY MQ PDB//\* MXG.DAILY.UOW - DAILY UOW PDB//\* MXG.DAILY.DATASETS - DAILY DATASETS PDB//\* MXG.DAILY.UOWSPIN - DAILY UOW SPIN//\* MXG.WEEKLY.PDB - WEEKLY UOW SPIN//\* MXG.MONTHLY.PDB - MONTHLY UOW SPIN//\* MXG.TREND.PDB - TREND PDB//\* YOUR.DAILY.SMF.ALL - ALL SMF RECORDS FOR ARCHIVAL//\* YOUR.DAILY.SMF.CICS - SMF 110.1 RECORDS FOR CICSTRAN//\* YOUR.DAILY.SMF.DB2 - SMF 101/102 FOR DB2//\* YOUR.DAILY.SMF.IO - I/O RECORDS FOR ANALDSAT//\* YOUR.DAILY.SMF.MQ - I/O RECORDS FOR ANALDSAT//\* YOUR.DAILY.SMF.SPLITPDB - ALL OTHER SMF RECORDS FOR "SPLIT PDB"//SYSIN DD \* DEFINE GENERATIONDATAGROUP - (NAME(MXG.DAILY.PDB) LIMIT(255) - NOEMPTY ) DEFINE GENERATIONDATAGROUP - (NAME(MXG.DAILY.SPIN) LIMIT(255) - NOEMPTY ) DEFINE GENERATIONDATAGROUP - (NAME(MXG.DAILY.IOPDB) LIMIT(255) - NOEMPTY ) DEFINE GENERATIONDATAGROUP - (NAME(MXG.DAILY.DB2ACCT) LIMIT(255) - NOEMPTY ) DEFINE GENERATIONDATAGROUP - (NAME(MXG.DAILY.CICSTRAN) LIMIT(255) - NOEMPTY ) DEFINE GENERATIONDATAGROUP - (NAME(MXG.DAILY.MQPDB) LIMIT(255) - NOEMPTY ) DEFINE GENERATIONDATAGROUP - (NAME(MXG.DAILY.CICS) LIMIT(255) - NOEMPTY ) DEFINE GENERATIONDATAGROUP - (NAME(MXG.DAILY.UOWSPIN) LIMIT(255) - NOEMPTY ) DEFINE GENERATIONDATAGROUP - (NAME(MXG.DAILY.DATASETS) LIMIT(255) - NOEMPTY ) DEFINE GENERATIONDATAGROUP - (NAME(MXG.DAILY.UOW) LIMIT(255) - NOEMPTY ) DEFINE GENERATIONDATAGROUP - (NAME(MXG.WEEKLY.PDB) LIMIT(255) - NOEMPTY ) DEFINE GENERATIONDATAGROUP - (NAME(MXG.MONTHLY.PDB) LIMIT(255) - NOEMPTY ) DEFINE GENERATIONDATAGROUP - (NAME(MXG.TREND.PDB) LIMIT(255) - NOEMPTY ) DEFINE GENERATIONDATAGROUP - (NAME(YOUR.DAILY.SMF.ALL) LIMIT(255) - NOEMPTY ) DEFINE GENERATIONDATAGROUP - (NAME(YOUR.DAILY.SMF.CICS) LIMIT(255) - NOEMPTY ) DEFINE GENERATIONDATAGROUP - (NAME(YOUR.DAILY.SMF.DB2) LIMIT(255) - NOEMPTY ) DEFINE GENERATIONDATAGROUP - (NAME(YOUR.DAILY.SMF.IO) LIMIT(255) - NOEMPTY ) DEFINE GENERATIONDATAGROUP - (NAME(YOUR.DAILY.SMF.MQ) LIMIT(255) - NOEMPTY ) DEFINE GENERATIONDATAGROUP - (NAME(YOUR.DAILY.SMF.SPLITPDB) LIMIT(255) - NOEMPTY ) |

So how many generations of each should you keep? It is up to you. In some instances there can be auditing requirements to keep data for long periods where accounting information is involved. In others there may be no such requirements and it will be a technical answer around how data you can afford to keep. My take on it:

 Daily datasets (excluding CICS and DB2 detail data) – 14-30 days

 Daily CICS/DB2 detail – 10 days

 Weekly datasets – 255 generations (a bit less than 5 years)

 Monthly datasets – 255 generations (assuming you choose to run monthly jobs)

 Trend dataset – 255 generations generally updated weekly but is highly summarized

Once you have defined the GDGs you can then build the JCL to execute the daily run:

## JCL for DAILY Job with dumped SMF data

|  |
| --- |
| //BUILDPDB EXEC MXGWPS or//BUILDPDB EXEC MXGSAS//SMF DD DSN=YOUR.DAILY.SMF,DISP=SHR//CICSTRAN DD DSN=MXG.CICSTRAN(+1),DISP=(,CATLG,DELETE),UNIT=TAPE//PDB DD DSN=MXG.DAILY.PDB(+1).DISP=(,CATLG,DELETE),// SPACE=(CYL,(1000,1000),RLSE)//SPIN DD DSN=MXG.DAILY.PDB(+1).DISP=(,CATLG,DELETE),// SPACE=(CYL,(10,10),RLSE)//SPININ DD DSN=MXG.DAILY.PDB(+0).DISP=SHR//SYSIN DD \*%LET SPININ=SPININ;%utilbldp( buildpdb=yes, useradd=, /\* any record types you want to add to the PDB \*/ suppress=, /\* any record types normally read by BUILDPDB to be suppressed \*/ outfile=instream);%bldsmpdb( runday=pdb, weekstrt=mon, /\* the day of the week your week starts on \*/ buildpdb=instream); |

If your DB2 or CICS data are compressed, you will want to assemble EXITCICS and add a STEPLIB for SAS (not for WPS) pointing to the LOAD library containing that LOAD module. While SAS and MXG will read and de-compress the data without the EXIT it is a very expensive process in terms of CPU time on zOS.

This job will read the SMF data and create the full PDB. There are many other options you could specify on UTILBLDP or BLDSMPDB but this is the basic structure of the job. But what if you want to read the data from a LOGSTREAM?

## JCL for Daily Job Reading LOGSTREAM

|  |
| --- |
| //BUILDPDB EXEC MXGWPS or//BUILDPDB EXEC MXGSAS//LOGGER DD DSN=YOUR.LOGSTREAM,DISP=SHR//CICSTRAN DD DSN=MXG.CICSTRAN(+1),DISP=(,CATLG,DELETE),UNIT=TAPE//PDB DD DSN=MXG.DAILY.PDB(+1).DISP=(,CATLG,DELETE),// SPACE=(CYL,(1000,1000),RLSE)//SPIN DD DSN=MXG.DAILY.PDB(+1).DISP=(,CATLG,DELETE),// SPACE=(CYL,(10,10),RLSE)//SPININ DD DSN=MXG.DAILY.PDB(+0).DISP=SHR//SYSIN DD \*%LET SPININ=SPININ;%LET MXGSMFREAD=LOGGER;%LET MACFILE=%QUOTE( IF DATEPART(SMFTIME)=TODAY()-1; /\* READ ONLY YESTERDAYS DATA \*/);%utilbldp( buildpdb=yes, useradd=, /\* any record types you want to add to the PDB \*/ suppress=, /\* any record types normally read by BUILDPDB to be suppressed \*/ outfile=instream);%bldsmpdb( runday=pdb, /\* required for GDG use \*/ weekstrt=mon, /\* the day of the week your week starts on \*/ buildpdb=instream); |

Three things change – the SMF DD becomes LOGGER, ‘%LET MXGSMFREAD=LOGGER;’ is inserted in the SYSIN ahead of UTILBLDP, and INFILE is set to read only data from yesterday with IF DATEPART(SMFTIME)=TODAY()-1;.

## JCL for Weekly Job

The weekly job consolidates the data for the last 7 days into a single SAS data library. By default, the WEEKBLD member copies only those datasets normally in the PDB and ignores any that you have added unless you manually add them. It is also dependent on the SORT order of the datasets and will ABEND if you have resorted any of the PDB datasets. BLDSMPDB does not share those characteristics and will automatically find any new datasets you may have added to the PDB but will also allow you to pick and choose the datasets that will be copied from the daily datasets to the weekly as well as allowing you to pick and choose whether to use the sort order or not. A word on sort orders – the sort orders of the datasets in the PDB are largely there to allow for the removal of duplicate data in the input datasets and are usually not very functional for reporting. In almost all cases, you will be sorting the data before generating a report and in some cases summarizing it as well. Because of that fact, preserving the sort orders in the weekly/monthly datasets has never been a major concern for me. In fact, suppressing the sort order is the default in BLDSMPDB as it prevents ABENDs when a new release changes the sort order of a dataset which does happen from time to time.

|  |
| --- |
| //WEEKBLD EXEC MXGSAS or//WEEKBLD EXEC MXGWPS//MON DD DSN=MXG.DAILY.PDB(-6).DISP=SHR//TUE DD DSN=MXG.DAILY.PDB(-5).DISP=SHR//WED DD DSN=MXG.DAILY.PDB(-4).DISP=SHR//THU DD DSN=MXG.DAILY.PDB(-3).DISP=SHR//FRI DD DSN=MXG.DAILY.PDB(-2).DISP=SHR//SAT DD DSN=MXG.DAILY.PDB(-1).DISP=SHR//SUN DD DSN=MXG.DAILY.PDB(-0).DISP=SHR//WEEK DD DSN=MXG.WEEKLY.PDB(+1),DISP=(,CATLG,DELETE),// SPACE=(CYL,(1000,1000),RLSE)//TREND DD DSN=MXG.TREND(+1),DISP=(,CATLG,DELETE),// SPACE=(CYL,(1000,1000),RLSE)//TRENDIN DD DSN=MXG.TREND(+0),DISP=SHR//SYSIN DD \*%LET TRENDOLD=TRENDIN;%Bldsmpdb(runday=no, runweek=yes, runmnth=no, weektape=no,  sortedby=no,  runtrnd=weekly, rollweek=no, weekstrt=mon, autotrnd=trndcec trndcelp trndcicx trnddb2a trnddb2p trnddbss trndrmfi trnd72go trndtmnt trndtalo, weekkeep=\_ALL\_, /\* OR LIST THE DATASETS TO KEEP \*/ weekDROP=spin: spun: /\* DATASETS TO DROP \*/);%include sourclib(any TRND members you have added or report programs); |

So what does this job do? First, it looks at the MON dataset and finds all of the SAS members contained in that SAS data library and makes a list of them. Then it compares that list to the WEEKKEEP and WEEKDROP parameters removing any that you don’t want to keep. Then it executes a series of data steps of the form:

 DATA WEEK.dataset;

 SET MON.dataset TUE.dataset WED.dataset THU.dataset FRI.dataset SAT.dataset SUN.dataset;

If you specified SORTEDBY=YES it will add a BY statement using the BY variables it found in the dataset in MON. If there is no BY attached to the dataset it is ignored.

After running all of the datasets it goes into TREND processing using the members shown above (the defaults) for AUTOTRND. If you are creating all of these then you do not need this parameter but if, for example you are not creating DB2 accounting data you should remove TRNDDB2A and TRNDDB2P from the list.

## JCL for Monthly Job - disk

To run a monthly job or not to run a monthly job? A good question to ask. If you are doing chargeback or accounting or some monthly reporting then the answer is most likely yes but if you don’t have those requirements it may not be necessary and may just chew up resources for no especially good reason. In any case, you almost certainly don’t want to keep all of the data. If you really need it later it is much faster in most cases to go back to the weekly.

|  |
| --- |
| //BUILDPDB EXEC MXGWPS//MON DD DSN=MXG.DAILY.PDB(-0).DISP=SHR//TUE DD DSN=MXG.DAILY.PDB(-1).DISP=SHR//WED DD DSN=MXG.DAILY.PDB(-2).DISP=SHR//THU DD DSN=MXG.DAILY.PDB(-3).DISP=SHR//FRI DD DSN=MXG.DAILY.PDB(-4).DISP=SHR//SAT DD DSN=MXG.DAILY.PDB(-5).DISP=SHR//SUN DD DSN=MXG.DAILY.PDB(-6).DISP=SHR//WEEK1 DD DSN=MXG.WEEKLY.PDB(-0),DISP=SHR//WEEK2 DD DSN=MXG.WEEKLY.PDB(-1),DISP=SHR//WEEK3 DD DSN=MXG.WEEKLY.PDB(-2),DISP=SHR//WEEK4 DD DSN=MXG.WEEKLY.PDB(-3),DISP=SHR//WEEK5 DD DSN=MXG.WEEKLY.PDB(-4),DISP=SHR//MONTH DD DSN=MXG.MONTHLY.PDB(+1),DISP=(,CATLG,DELETE),// SPACE=(CYL,(1000,1000),RLSE)//SYSIN DD \*%bldsmpdb(runday=no, runweek=NO, runmnth=YES, runtrnd=weekly, weekstrt=mon, MNTHTAPE=NO, /\* ASSUMES MONTH IS ON DISK \*/ MNTHkeep=\_ALL\_, /\* OR LIST THE DATASETS TO KEEP \*/ MNTHDROP=spin: spun: /\* DATASETS TO DROP \*/); |

In the case of the monthly job, BLDSMPDB looks at WEEK1 to see what datasets exist and bases the selection on that and the dataset listed in the MNTHKEEP and MNTHDROP parameters. In addition, it has to determine what data to keep based on ZDATE (zee date zee OBS was created.) For any given month the correct ZDATE values are from the second of that month to the first of the following month. With a month being composed of 28-31 days, the data could be in the last 5 weekly datasets or in some combination of those and the last 7 daily datasets depending on the day of the week on which WEEks start and the day of the week on which the 1st of the month falls. If, for example, the 1st falls on Wednesday but the week starts on Monday, the month would be the data from Tue and Wed this week and the data from WEEK1-WEEK5 where the ZDATE value falls into the correct range. Since in most cases on zOS the JCL for production jobs must be static and the relative GDG numbers may not always be in the correct sequence, the code figure out where the current week started and will bypass any records with a ZDATE prior to the start of the current week.

This copy of the job assumes that the monthly database will fit on disk. If it is too large to fit then some radical steps must be taken. When SAS writes to a sequential dataset be it on tape or disk, it first has to read all of the datasets currently on the tape to be sure the dataset does not already exist then adds the dataset to the end. On physical tape this results in a rewind and read for every dataset and if the size of the volume is exceeded many mounts and remounts. MXG circumvents this by writing the individual datasets to disk then treating the output tape as a file rather than a SAS dataset and appends each individual SAS dataset to the tape. The JCL for that method is:

## JCL for Monthly Job - Tape

|  |
| --- |
| //BUILDPDB EXEC MXGWPS//MON DD DSN=MXG.DAILY.PDB(-0).DISP=SHR//TUE DD DSN=MXG.DAILY.PDB(-1).DISP=SHR//WED DD DSN=MXG.DAILY.PDB(-2).DISP=SHR//THU DD DSN=MXG.DAILY.PDB(-3).DISP=SHR//FRI DD DSN=MXG.DAILY.PDB(-4).DISP=SHR//SAT DD DSN=MXG.DAILY.PDB(-5).DISP=SHR//SUN DD DSN=MXG.DAILY.PDB(-6).DISP=SHR//WEEK1 DD DSN=MXG.WEEKLY.PDB(-0),DISP=SHR//WEEK2 DD DSN=MXG.WEEKLY.PDB(-1),DISP=SHR//WEEK3 DD DSN=MXG.WEEKLY.PDB(-2),DISP=SHR//WEEK4 DD DSN=MXG.WEEKLY.PDB(-3),DISP=SHR//WEEK5 DD DSN=MXG.WEEKLY.PDB(-4),DISP=SHR//TAPETEMP DD UNIT=(SYSDA,4),SPACE=(CYL,(1500,1500))//DUMMY DD SPACE=(CYL,5)//MONTH DD DSN=MXG.MONTHLY.PDB(+1),DISP=(,CATLG,DELETE),// UNIT=3590 **<-whatever esoteric works for you**//SYSIN DD \*%bldsmpdb(runday=no, runweek=NO, runmnth=YES, runtrnd=weekly, weekstrt=mon, MNTHTAPE=YES, /\* ASSUMES MONTH IS ON TAPE \*/ MNTHkeep=\_ALL\_, /\* OR LIST THE DATASETS TO KEEP \*/ MNTHDROP=spin: spun: /\* DATASETS TO DROP \*/); |

# Chapter 12: Running BUILDPDB on ASCII

For all of the ‘jobs’ on ASCII platforms it is recommended that you use the AUTOALOC feature and use the FTP INFILE method to get the SMF data from zOS.

|  |
| --- |
| Daily JOB on ASCII Platform |
| FILENAME SMF FTP ("'your.daily.SMFDATA'" ) USER='userid' HOST='ftp address' DEBUG S370VS RCMD='SITE RDW' LRECL=32760 PASS='password';%utilbldp( buildpdb=yes, useradd=, /\* any record types you want to add to the PDB \*/ suppress=, /\* any record types normally read by BUILDPDB to be suppressed \*/ outfile=instream);%bldsmpdb( autoaloc=yes, basedir=c:\mxg, /\* where to store the data \*/ datefmt=yymmdd, /\* the format of the date in directory names \*/ day2keep=dd, /\* the number of daily databases to keep \*/ wek2keep=ww, /\* the number of weekly databases to keep \*/ cicskeep=cc, /\* the number of CICSTRAN databases to keep \*/ db2keep=xx, /\* the number of DB2ACCT databases to keep \*/ mth2keep=mm, /\* the number of monthly databases to keep \*/ runday=yes,  runweek=yes, /\* could also be WTD for week to date \*/ runmnth=yes, /\* could also be MTD for month-to-date \*/ runtrnd=weekly, /\* could also be daily \*/ weekstrt=mon, /\* the day of the week your week starts on \*/ autotrnd=trndcec trndcelp trnddb2a trnddb2b trnddbss trndrmfi trnd72go trndtmnt trndtalo, weekkeep=\_all\_, weekdrop=spin: spun:, mnthkeep=\_all\_, mnthdrop=spin: spun:, buildpdb=instream); |

There are some differences in the coding of the jobs when you move from zOS to ASCII. The biggest difference being there is only ONE job! BLDSMPDB on ASCII will do it all in a single job. When it is time to run weekly/monthly/trending it is run as part of the same job. It is worth some time here to review the parameters and what they all mean.

First and very important is the FILENAME statement that allocates the SMF dataset. This can be a tape or disk dataset on your zOS system it makes no difference which so long as the USERID you are using has READ access to the data, FTP access, and can mount a tape. This most likely means it will not be your TSO userid.

The UTILBLDP invocation is the same as it would be on zOS. You can add user records, suppress records, all of the other things that are possible with UTILBLDP.

The invocation of BLDSMPDB is however very different.

It starts out with the parameters specifying the locations and length of time to keep the various datasets that will be created;

|  |  |
| --- | --- |
| **Parameter** | **Usage** |
| AUTOALOC=YES | Use VMXGALOC to allocate directories for the output performance data bases |
| BASEDIR=C:\MXG | The base directory that is used. This can be on any local or networked drive the only caveat is that it must be available to MXG for WRITE |
| DATEFMT=YYMMDD | Hindsight is always 20/20. The default value for the DATEFMT is DATE which gives you directory names like D26OCT14 – not terribly functional as a sort order when you list the directories. Once it was done it was too late to change and disrupt people already using it. I find the YYMMDD format more ‘user friendly’ as it puts the directories in chronological order when you list them. NOTE that the date in the directory name reflects the date of the data contained in that directory so D141026 has the data for Oct 26, 2014 and W141026 the weekly data starting with Oct 26, 2014. |
| DAY2KEEP=xxWEK2KEEP=xxCICSKEEP=xxDB2KEEP=xxMTH2KEEP=xx | Xx is the number of generations of each sort of directory to retain. VMXGALOC keeps track and when the lifetime based on number of days has passed the directories are removed along with their contents. The constructed directories look like:Ddate – DAILYSdate – SPINCICSdate – CICSDB2date –DB2Wdate – WEEKLYMdate – MONTHLYTdate – TRENDSpin directories use the DAY2KEEP value and TREND directories are dependent on the value specified for RUNTRND be it WEEKLY or DAILY.If you specify CICSKEEP=0 or DB2KEEP=0 those directories are not constructed. The DB2ACCT dataset will automatically go to the PDB LIBNAME but if you want CICS to go there you must modify using mackeep. |

Next we tell it what to run.

|  |  |
| --- | --- |
| **Parameter** | **Usage** |
| RUNDAY=YES | Run the daily job. This differs from zOS in that on zOS you specify RUNDAY=PDB which puts the data in the PDB DD while RUNDAY=YES puts the data in the LIBNAME pointed to by the day of the week – MON TUE WED THU FRI SAT SUN |
| RUNWEEK=YES | Run the weekly job on the day specified by the WEEKSTRT parameter. Alternatively, you can specify RUNWEEK=WTD and a week to date weekly dataset will be constructed as part of the daily job. |
| WEEKSTRT=MON | The day of the week on which your week begins. It can be any day of the week you choose. |
| RUNTRND=WEEKLY | Run the TREND code weekly. Or you can choose DAILY and keep the TREND database very current. |
| RUNMNTH=YES | Run the monthly of the 1st day of each month or specify MTD and the monthly will be updated on a daily basis. |
| FIRSTRUN=YES/NO | NO is the default but the first time you run this you need to specify YES. That will build all of the directories and populate them with 0 OBS datasets so that weekly/monthly processes will run without errors. |
| WEEKKEEP | \_ALL\_ says to copy all of the datasets in MON to the WEEKLY PDB. You can specify specific datasets to keep or leave it at \_ALL\_ and use WEEKDROP to drop those you don’t want to keep. Which ever list is shorter is the rule of thumb. |
| WEEKDROP | SPIN: SPUN: datasets really don’t add much to the WEEKLY PDB and by default get dropped. You can add as many other datasets as you want to this list and they will not be propagated into the WEEKLY PDB. |
| MNTHKEEP | \_ALL\_ says to copy all of the datasets in WEEK1 to the MONTHLY PDB. You can specify specific datasets to keep or leave it at \_ALL\_ and use MNTHDROP to drop those you don’t want to keep. Which ever list is shorter is the rule of thumb. |
| MNTHDROP | SPIN: SPUN: datasets really don’t add much to the MONTHLY PDB and by default get dropped. You can add as many other datasets as you want to this list and they will not be propagated into the MONTHLY PDB. |
| BUILDPDB=INSTREAM | This simply tells BLDSMPDB that the code to drive the BUILDPDB process is in the FILENAME INSTREAM that was created by UTILBLDP. It can also point to a tailored member of SOURLIB such as MYPDB but INSTREAM is the more common usage. |

Once the jobstream is constructed it is now a matter of scheduling. If you have scheduling software that crosses the boundaries between zOS and ASCII and can start a job based on the creation of the SMF data, this may be the thing to do. But, both Windows and LINUX have time based scheduling facilities that will work just fine.

# Chapter 13: What about non-SMF sources?

There are some non-SMF data sources that it is common for people to process along with the SMF data. The most likely candidates are data from the tape management catalog and data from the DCOLLECT utility containing information on all of the DASD datasets and volumes. BLDSMPDB has these data sources added as optional add-ons to the daily BUILDPDB process (and by extension into the weekly and monthly processing.

|  |  |
| --- | --- |
| Parameter | Usage |
| TMC=TMC | Read the CA tape management catalog – if blank not used |
| RMM=EDGHSKP | Read the IBM RMM tape catalog  |
| CONTROLT=CONTROLT | Read the BMC Control T catalog |
| DCOLLECT=DCOLLECT | Read the output of DCOLLECT |
| DAILYDSN=TMC or RMM or CONTROLT  | Run DAILYDSN (collects the data on all datasets into the PDB as DATASETS) using the indicated tape management data |

This can be done on both zOS and ASCII systems but the DCOLLECT job MUST be run on zOS. The JCL to run DCOLLECT is here and also in the SOURCLIB as JCLDCOL. Items in **RED** need changing.

|  |
| --- |
| DCOLLECT JCL |
| //JOBCARD JOB WHATEVER WORKS FOR YOU//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//\*\* RUN DCOLLECT TO CAPTURE DASD STATISTICS - ALWAYS RUN \*\*//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//STEP1 EXEC PGM=IDCAMS//SYSPRINT DD SYSOUT=\*//DCOLLECT DD DSN=**MY.DCOLLECT.DATA**,SPACE=(CYL,(20,25)),// LRECL=32756,RECFM=VB,DSORG=PS, <- REQUIRED LRECL// DISP=(,CATLG,DELETE)//MCDS DD DSN=**HSM.MCDS,DISP=SHR <- YOUR MIGRATION CATALOG**//BCDS DD DSN=**HSM.BCDS,DISP=SHR <- YOUR BACKUP CATALOG**//SYSIN DD \* DCOLLECT OUTFILE(DCOLLECT) VOLUMES(\*) MIGRATEDATA BACKUPDATA - CAPPLANDATA//\* |

In the case of zOS the rest is a matter of adding DDNAMEs for DCOLLECT and the tape catalog of choice to the JCL for running the daily job. BLDSMPDB will take care of the rest.

On ASCII you must assign FILENAME statements to capture the data via FTP just as you do with SMF but with some differences.

|  |
| --- |
| ASCII FILENAME Statements for DCOLLECT and TMC |
| FILENAME DCOLLECT FTP ("'your.dcollect.data'")  USER='userid' HOST='ftp adress' DEBUG  s370vs PASS='password' rcmd='site rdw' lrecl=932;FILENAME TMC FTP ("'your.TMC.data'")  USER='userid' HOST='ftp adress' DEBUG  PASSS='password' recfm=f lrecl=340; |

There are many other non-SMF sources of data. The techniques for reading them are basically the same. Using DCOLLECT as an example outside of BLDSMPDB:

|  |
| --- |
| Read DCOLLECT data on zOS |
| //DCOLLECT DD DSN=**MY.DCOLLECT.DATA**,SPACE=(CYL,(20,25)),// LRECL=32756,RECFM=VB,DSORG=PS, <- REQUIRED LRECL// DISP=(,CATLG,DELETE)//MCDS DD DSN=**HSM.MCDS,DISP=SHR <- YOUR MIGRATION CATALOG**//BCDS DD DSN=**HSM.BCDS,DISP=SHR <- YOUR BACKUP CATALOG**//SYSIN DD \* DCOLLECT OUTFILE(DCOLLECT) VOLUMES(\*) MIGRATEDATA BACKUPDATA - CAPPLANDATA//MXG EXEC MXGSAS//DCOLLECT DD DSN=MY.DCOLLECT.DATA,DISP=SHR//PDB DD DSN=MY.DCOLLECT.PDB(+1),DISP=(,CATLG,DELETE),SPACE=(CYL,(400,400))//SYSIN DD \*%INCLUDE SOURCLIB(TYPSDCOL); |

|  |
| --- |
| Read DCOLLECT data on ASCII |
| FILENAME DCOLLECT FTP ("'your.dcollect.data'")  USER='userid' HOST='ftp adress' DEBUG  s370vs PASS='password' rcmd='site rdw' lrecl=932;**%VMXGALOC(BASEDIR=..); < your normal VMXGALOC parameters**%INCLUDE SOURCLIB(TYPSDCOL); |

# Chapter 14: GAAAAAH!!!! It ABENDed!!! What Now?

Not to worry. In most cases all you have to do is rerun the job. The most likely problem on zOS is an out of space condition on one of the datasets.