MWSUG 2016 - Paper BB26

Implementing a Medicaid Claims Processing System using SAS® software A Novel Implementation of BASE SAS® Using Limited Resources in an Effective Manner

Stephen C. DeVoyd, Ohio Department of Developmental Disabilities, Columbus, OH

ABSTRACT

When you think of Base SAS® what comes to mind? A processor of huge amounts of data which can be presented in a myriad of ways? Charts and graphs, detailed spreadsheets? Perhaps you think of advanced statistical measures?

The Ohio Department of Developmental Disabilities (ODoDD) utilizes base SAS® in a novel or unconventional way. The purpose of this abstract will be to explain how ODoDD utilizes SAS® to support claims processing for over 40,000 individuals served by almost 15,000 Medicaid waiver providers for each of the 4 Medicaid Waivers administered by ODoDD.

Each week over 6,000 claims files are collected electronically at the DODD secured website. These files are processed by a series of complex programs which evaluate an average of 600,000 claims weekly using data from SAS® datasets which are sized from a few records to over 200,000,000 records. After ODoDD completes initial editing the data, the claims are sent electronically to the Ohio Department of Medicaid (ODM) for review and adjudication. The resulting adjudication file returned to ODoDD contains approved and denied claims. The claims marked to be paid are then processed by ODoDD's SAS® payment processing programs. These SAS® payment programs create vouchers for Medicaid providers for over 27,000,000 claims submitted annually. This volume represents over $1,500,000,000 in annual reimbursements to the provider community.

The process outlined above is accomplished by using BASE SAS® software leveraging minimal SAS® Programming FTE's to support this dynamic and and ever-changing environment.

This paper and presentation will include:

- Overall System Architecture including SAS® instances running in virtual server environments
- ODoDD's use of SAS®, SAS® -ODS and SAS® -SQL.
- How ODoDD leverages SAS® to accomplish tasks in an efficient manner.
- Issues and lessons learned by ODoDD, including dealing with size and complexities and what ODoDD has done to address them.

INTRODUCTION

In 1985, the Ohio Department Developmental Disabilities (ODoDD), then called Ohio Department of Mental Retardation and Developmental Disabilities (ODMRDD), was split from the Ohio Department of Mental Health. The decision to use SAS® to process Medicaid claims for providers of service to ODMRDD individuals was made. At the time there was one main programmer to design, develop and maintain the source code for the Medicaid Billing System and SAS® was selected due the ease and flexibility of coding.

SYSTEM OVERVIEW

ODoDD has transformed the overall system architecture tremendously from the 1990's to today. Figure 1 illustrates the evolution of the system supporting the MBS system:

Service Year	1992	1996	2009	2015
Platform	IBM Mainframe (to 1993)	VAX (1993 - 2009)	Windows 64-bit Server 2008 (2009 - 2016)	Windows Server 2012 R2 (2016 -)
SAS® Version	4.X-5.X	6.X	9.2	9.4
Job Scheduler	JCL	Command Files	Job Access and Management System (JAMS)	Job Access and Management System (JAMS)
Number Claims Processed	1,960,672	4,848,179	13,057,620	18,300,307
Full Time Staff	1	2-3	2-3	2
Reimbursed Amount	$52,632,557.29	$248,416,829.38	$1,016,675,019.65	$1,482,225,954.44

Figure 1 - History of ODoDD System

As can be seen, the number of transactions processed annually have increased by over 9000%, but the staffing has not changed significantly over that time. There have been contract employees through the years that have aided in the development of larger initiatives, the core number of people supporting the system have remained constant.

The Medicaid Billing System (MBS) system consists of nearly 160 program and over 80,000 lines of executable SAS® code. The system is designed as a weekly batch system with monthly, quarterly and annual reports. The Figure 2 illustrates the weekly MBS processing flow:

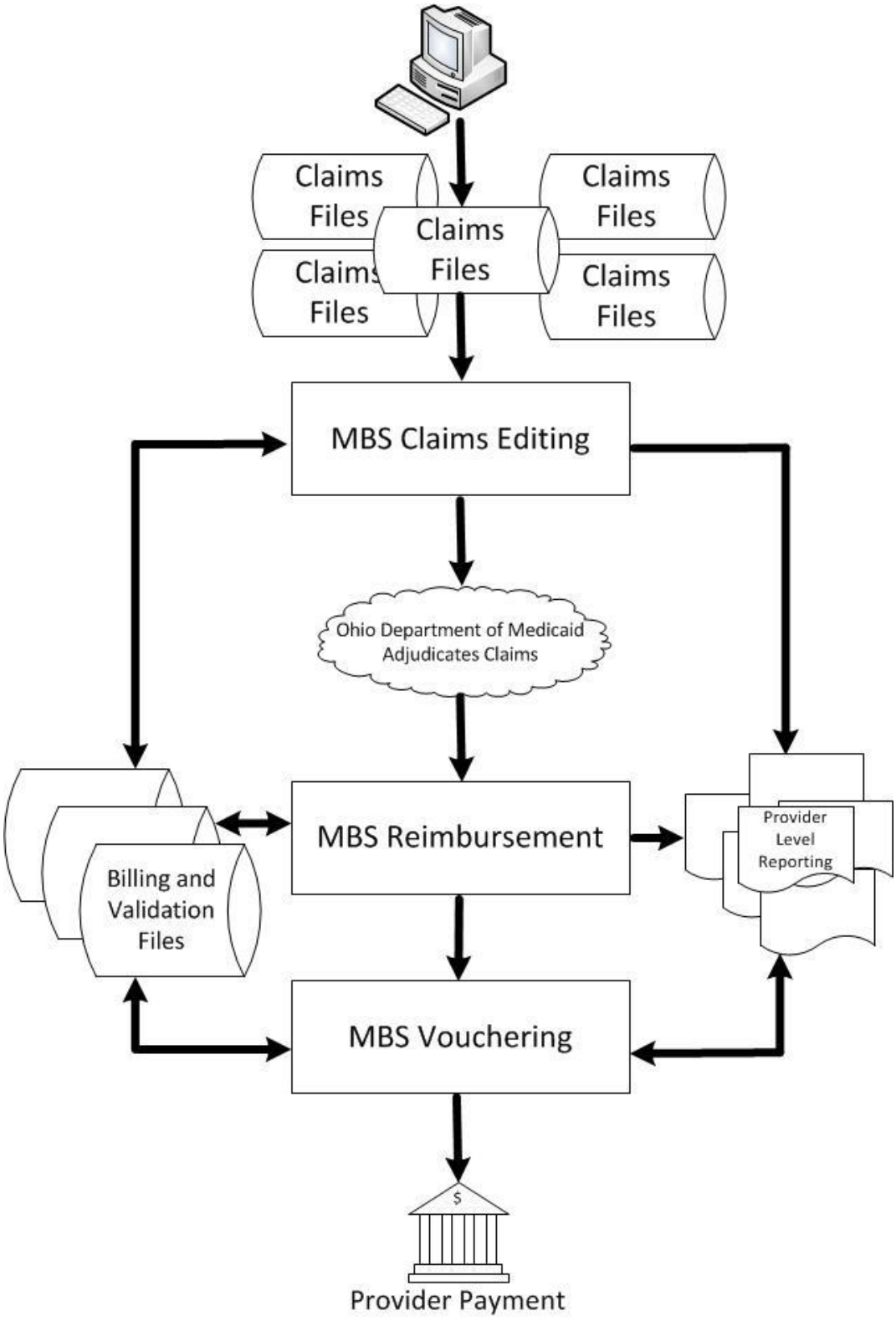


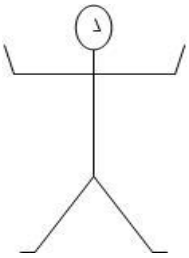
Figure 2 - MBS Weekly Processing Flow

PRESENTATION FORMAT

As this is the year of the 2016 Olympics, the paper will be structured similar to the games. There are three (3) main topics…or events in Olympic terms. Each of the topics will have a standard by which success is judged.

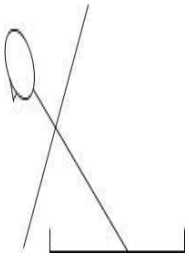
The Events

For each event one issue/problem will be presented. Each issue was approached from several different directions. The approaches will be evaluated based on the criteria that is noted.



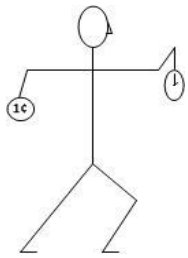
Performance

- Gains were made either in clock time or disk savings.
- Cost of the performance solution.
- Time to implement.



Flexibility

- Ease at which changes were designed and implemented.
- Reusability of the code either in logic or method.
- Easily understood.



Savings

- Processes and procedures to save time in development.
- Efficient utilization of available tools and equipment.
- Proper utilization of the skills of the limited staff

The Medals

The medal that will be awarded are based on the success of the solution that was tried:

Bronze Medal Winner

Bronze Medal is defined as a solution that was tried but met with limited success or no discernable differences were noticed.

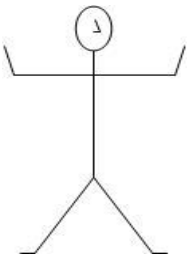
Silver Medal Winner

The winner of the Silver Medal is defined as a solution that worked but still had some shortfalls to total success.

Gold Medal Winner

The Gold Medal is defined as a solution that satisfied the problem being resolved and met all of the performance criteria for the event.

FIRST EVENT - PERFORMANCE



Performance

- Gains were made either in clock time or disk savings.
- Cost of the performance solution.
- Time to implement.

ISSUE TO RESOLVE

From week to week the execution clock time of the Medicaid Billing System (MBS) is fairly static. The number of claims that are submitted by providers ranges between 350,000 to 800,000 per week but the total clock time for the jobs is not based on the number new claims in the door. The job driver is the amount of historical claims that the new incoming claims need to be evaluated against to ensure that the claim has not already been paid, falls within the budget limitations based on the unique needs of each individual and to properly discern a duplicate claim from the adjustment of a previously paid claim.

The total runtime of the batch jobs should stay within 20% of the average runtime of the job, based on activity on the server outside the batch processing. There was an observation made that the runtime of all jobs were increasing across the board. The jobs steps that manipulated large amounts of data through PROC SORT steps, DATA/SET steps and DATA/MERGE were increasing both clock time and CPU time at an alarming rate. Table 1 illustrates examples of several long running steps in one of the jobs.

Date	File Backup using DATA/SET			Merge of two large datasets			PROC SORT of large dataset		
	Clock Time	CPU time	# of Rcds	Clock Time	CPU Time	# of Rcds	Clock Time	CPU Time	# of Rcds
2/16/2016	**06:44.7**	**43.91**	12,129,144	**06:53.8**	**14.49**	12,129,144	**07:10.6**	**12.65**	12,129,144
2/8/2016	04:38.5	31.84	12,101,901	**04:34.9**	**11.37**	12,101,901	**04:28.7**	**12.41**	12,101,901
2/1/2016	**05:21.5**	**44.47**	12,073,484	00:07.9	7.84	12,073,484	00:09.8	9.36	12,073,484
1/25/2016	04:31.3	41.66	12,048,211	**04:19.7**	**1.54**	12,048,211	**02:03.0**	**9.64**	12,048,211
1/19/2016	**06:16.4**	**45.53**	12,029,741	**04:46.1**	**13.12**	12,029,741	00:08.7	8.36	12,029,741
1/11/2016	03:58.5	25.0	12,004,936	00:06.6	6.55	12,004,936	00:07.3	6.94	12,004,936
1/4/2016	04:06.1	23.9	11,982,749	**04:05.4**	**7.25**	11,982,749	00:06.3	6.05	11,982,749
12/28/2015	04:14.6	21.12	11,972,841	00:05.9	5.75	11,972,841	00:06.1	5.94	11,972,841

Table 1 - Clock and Run Times of Long Running Steps

Using 12/28/2015 as the base to which the other weeks are compared, it can be seen that each week there was at least one step that took up to 50% or more than the 12/28 standard.

There were a lot of ideas and potential resolutions that were tried that did not make the podium including:

1. There were some indications that not all memory was being released after the job completed.

Rebooted server and found no significant effect.

2. There was some consideration given that third party software's installed on the serve may be having some impact. Given the SAS® Server has no access outside the State of Ohio IT network and no outside entity has access to it we were able to selectively remove these software's to rule out impact. After going through this exercise it was determined there was no significant change by removing these add-on's
3. All of the jobs and code were validated to ensure that no significant changes had been made. There were none.
4. Validated with the State of Ohio IT group that no significant changes were made to the network or servers. There was none.

BRONZE MEDAL

ADJUSTING THE TOTAL MEMORY ON THE SERVER

The Bronze Medal went to extending the server memory from 42 gigabytes to 65. This change was made dynamically on the virtual SAS® server. The idea was if there was insufficient memory to load the enough of the temporary files that a PROC SORT, for example, that having more in memory could help clock time. The results of this were mixed. The processing did go to memory and more CPU time instead of Disk I/O as expected. Some of the steps were outside 15% better than standard, but most did not hit that threshold.

As taking an additional 22 gigabytes of memory is a high cost for limited effectiveness, so the memory was set back to the 42 gigabytes.

SILVER MEDAL

ADJUSTING THE SCATTER-READ/GATHER-WRITE AND THE NUMBER OF BUFFERS ALLOCATED FOR PROCESSING SAS® DATASETS

To help identify SAS® configuration options, a ticket was opened with the SAS® support center. There were two suggestions made:

- Validate the setting of the -SGIO System Option. This option improves I/O performance for SAS® I/O files (data sets, catalogs, indexes, utility files, and other I/O files) when the Server has a large amount of RAM. Scatter-read / gather-write bypasses intermediate buffer transfers between memory and disk.

  When SGIO is active, SAS® uses the number of buffers that are specified by the BUFNO system option to transfer data between disk and RAM. I/O performance usually improves as the value for the BUFNO increases.

- The BUFNO= system option is used by the SGIO System option. Using BUFNO= can improve

execution time by limiting the number of input/output operations that are required for a particular SAS® data set. However, the improvement in execution time comes at the expense of increased memory consumption.

The optimal value for buffer system options is dependent on your operating environment. Experiment with various buffers sizes to determine the optimal value for these system options.

- The BUFSIZE= system option is used by the SGIO System option. The buffer size is the amount of data that can be transferred from a single input/output operation to one buffer. The buffer size is a permanent attribute of the data set and is used when the data set is processed.

  A larger buffer size can improve execution time by reducing the number of times SAS® has to read from or write to the storage medium. However, the improvement in execution time comes at the expense of increased memory consumption.

  The optimal value for buffer system options is dependent on your operating environment. Experiment with various buffers sizes to determine the optimal value for these system options.

The default values set in the default SAS® system configuration file is:

- **-SGIO System Option is ON**

  Under the Windows operating environment, if the SGIO system option is set, the maximum number of bytes that can be processed in an I/O operation is 64MB.

- **BUFNO=1**

  This indicates that a 1 buffer is allocated for a 1K buffer size.

- **BUFSIZE=0**

  This value sets the buffer size to the default value in the operating environment.

To determine the optimal value of these, a series of tests were done. Three different steps from a long running program were selected. The default setting was set by a production run of the three steps. The SAS® system defaults were in place. The total clock time for the three steps was 27 minutes and 3.2 seconds. The same data on the same server was then run using the BUFNO=200 and BUFNO=300 with BUFSIZE=64K, BUFSIZE=128K and BUFSIZE=256K. Once the number of buffers or the buffer size got larger, the server ran into memory issues. Table 2 and Table 3 show the results of the test.

Short description of Work of step	Production Run 03/17/2016 BUFNO=1 SGIO=OFF (in seconds)		BUFNO=200 SGIO=ON (in seconds)					
	BUFSIZE=0	BUFSIZE=0	BUFSIZE=64K		BUFSIZE=128K		BUFSIZE=256K	
	Clock Time	CPU Time	Clock Time	CPU Time	Clock Time	CPU Time	Clock Time	CPU Time
Read/Write of Bill Data to create a file of 58M records	1134.0	110.8	257.0	96.7	248.2	94.6	234.4	93.7
Sort of 58M records	244.6	88.0	111.5	61.8	81.8	58.0	77.9	61.0
Data/Merge of 58M records with 52K records to Produce a 27M record file	244.6	30.0	74.0	16.8	58.7	16.5	48.0	63.0
Time in total Seconds	1623.2	228.8	442.5	175.3	388.7	169.1	360.3	217.7
	27 min 3.2 sec	3 min 48.8 sec	7 min 22.5 sec	2 min 55.3 sec	6 min 28.7 sec	2 min 49.1 sec	6 min 0.3 sec	3 min 37.7 sec

Table 2 - Clock and CPU Times of Long running steps with BUFNO=200

Short description of Work of step	Production Run 03/17/2016 BUFNO=1 SGIO=OFF (in seconds)		BUFNO=300 SGIO=ON (in seconds)					
	BUFSIZE=0	BUFSIZE=0	BUFSIZE=64K		BUFSIZE=128K		BUFSIZE=256K	
	Clock Time	CPU Time	Clock Time	CPU Time	Clock Time	CPU Time	Clock Time	CPU Time
Read/Write of Bill Data to create a file of 58M records	1134.0	110.8	278.4	278.1	93.9	196.2	93.9	93.7
Sort of 58M records	244.6	88.0	57.8	96.1	57.6	69.2	57.1	61.0
Data/Merge of 58M records with 52K records to Produce a 27M record file	244.6	30.0	16.8	54.3	16.4	44.7	16.6	16.8
Time in total Seconds	1623.2	228.8	353.0	428.5	167.9	310.1	167.6	171.5
	27 min 3.2 sec	3 min 48.8 sec	5 min 53.0 sec	7 min 8.5 sec	2 min 47.9 sec	5 min 10.1 sec	2 min 47.6 sec	2 min 51.5 sec

Table 3 - Clock and CPU Times of Long running steps with BUFNO=300

All the results were statistically significant with any of the settings outside of the defaults. The decision was made, with the recommendation of the infrastructure team, who was monitoring the memory of the server while these jobs were running was to use the BUFNO=200 and BUFSIZE=128K. These changes were made in the SAS® system configuration file.

The reason this solution was not the Gold Medal winner, is that not all jobs had performance gains. In the weekly processing, there are provider level reports created which had an increase of time of about 3000%. These jobs create between 8,000 and 12,000 very small text files. The overhead to create these files with the optimized BUFNO= and BUFSIZE= caused the jobs to run much slower. To resolve this, for jobs that had the increased clock time, the settings were set to –NOSGIO (to turn off Scatter/Gather) and BUFNO=0 to set it back to the server settings.

GOLD MEDAL

Upgrade Server from Windows 64-bit Server 2008 to Windows 2012 R2 Server

On a standard week, the provider claims files are picked up from the drop off folder by 4PM on Wednesday and the billing jobs are run on Thursday. The performance of the billing portion of the weekly schedule has historically been a focus as the file sent to our sister state agency needs to be received by Friday at 5PM. The claims data is also swept weekly over the weekend and sent to the ODoDD Business Intelligence group of the weekend to be consumed by the COGNOS® Reporting system.

When the system was running on the VAX machine, the elapsed time of the job was between 8 and 20 hours. When there were issues, it made it difficult to resolve before the processing had to be completed. The jobs were changed to create restart points by saving key files and if an issue were found, the code would be executed from the previous checkpoint. This made the code more complex to write and maintain.

Moving to a Windows 64-bit 2008 Server and upgrading to SAS® 9.1 greatly sped up processing. The elapsed time of the jobs were between 4 and 6 hours. The lowering of processing allowed more flexibility in resolving issues and made processing on a holiday week easier. After some performance tuning of the SAS® code, the processing time was averaging about 3 hours.

In the investigation on the performance issue, the infrastructure team found many different references to performance gain of the Microsoft 2012 Server. Some of the memory issues that the 2008 Server had were resolved in the 2012 Server. To test the gains, the test system was updated to the 2012 Server and the initial tests showed a significant difference. The steps that were memory and I/O intensive saw significant gains. At the same time, it was decided to upgrade from SAS® version 9.3 to 9.4.

Table 4 and 5 describe the production performance differences between the two platforms. The jobs were run with the same code and data. The differences were the operating system and the version of SAS®.

Step	Week 1 (in seconds)						Week 2 (in seconds)					
	Production with SAS® 9.3		New Server with SAS® 9.4		Difference		Production with SAS® 9.3		New Server with SAS® 9.4		Difference	
Step 1	2975	796	1964	858	66%	108%	3844	802	1754	838	46%	104%
Step 2	3492	776	2103	774	60%	100%	4647	759	1979	725	43%	96%
Step 3	810	150	479	194	59%	129%	586	149	500	235	85%	158%
Step 4	1279	234	719	259	56%	111%	1380	236	584	264	42%	112%
Step 5	183	65	108	66	59%	102%	135	63	108	74	80%	117%
Step 6	377	94	243	92	64%	98%	400	95	264	121	66%	127%
Reports	1016	868	982	849	97%	98%	964	810	807	763	84%	94%
Total	10132	2983	6598	3092	65%	104%	11956	2914	5996	3020	80%	103%
	2hr 48min	49min 43sec	1hr 50min	51min 32sec			3hr 19min	48min 34sec	1hr 39min	50min 20sec		

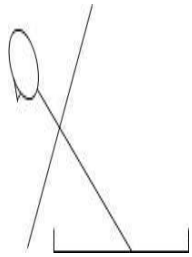
Table 4 - lock and CPU times of the 7 longest steps of the billing jobs for a two week parallel timeframe.

Step	Average in Seconds					
	Production with SAS® 9.3		New Server with SAS® 9.4		Difference	
Step 1	3410	799	1859	848	54%	106%
Step 2	4070	768	2041	750	50%	98%
Step 3	698	150	490	215	70%	143%
Step 4	1330	235	652	262	49%	111%
Step 5	159	64	108	70	68%	109%
Step 6	389	95	254	107	65%	113%
Reports	990	839	895	806	90%	96%
Total	11044	2949	6297	3056	57%	104%
	3hr 4 min	49min 9sec	1hr 45min	50min 56sec		

Table 5 - Average Clock and CPU times of the 7 longest steps of the billing jobs.

As can be seen, there are tremendous time gains in the upgraded server model. The reports job did not perform significantly different, but the gains in all the other jobs make this solution easily the Gold Medal Winner.

SECOND EVENT - FLEXIBILITY



Flexibility

- Ease at which changes were designed and implemented.
- Reusability of the code either in logic or method.
- Easily understood.

ISSUE TO RESOLVE

There is not one specific issue to address for Flexibility as there was with Performance. The contenders of this event that were selected are more approaches to a global issue…from performance…to SAS® functionality…to project implementation. Each of the winners and losers will have the issue that it was attempting to address and the success of the approach.

There were a lot of ideas and potential resolutions that were tried that did not make the podium including:

1. Issue: Documenting code

   All code needs to be commented to be understood over time. The comments can get out of hand by commenting every line. Only sections of logic that are complex or an exception for a specific reason is really necessary in a line by line documentation style. As the code ages, the many comments get less and less meaningful.

2. Issue: Managing variables in a SAS® program

   In a SAS® program that has multiple steps, there are can be many extraneous variables added to the data. The processing of large files with many extra variables can negatively affect performance. That being said, the use of DROP and KEEP statements can become a programmer's nightmare. When modifying code I have been caught by dropping a variable in the beginning of the program that, at the time is not relevant to the rest of the job, but is needed for the new section of code being added.

   I have found the DROP easier to code with, as the variable can just be removed from the DROP statement and the variable is kept. The KEEP statement has more impact to future enhancements…as all the KEEP statements between where the variable was initially dropped due to a KEEP statement and where the variable is needed further down need to be modified.

   It is good to keep lean when managing variables to ensure better performance….but my suggestion is to DROP through the code and then KEEP when saving the permanent version of the dataset.

3. Issue: Compressed Folders

   Compressed files have the potential for large savings of space especially with data with a lot of space. In an effort to save a step in the weekly processing, ODoDD compressed the backup folder. The idea was to compress the folder, to have all the files compressed as they were written. The performance of this was miserable. The amount of time to compress all the files after being written is much less than compressing as they are being written.

BRONZE MEDAL

Reporting Output – .XLSx vs. .TXT

ODoDD produces tens of thousands of reports weekly. The user community for the reports includes internal ODoDD central office staff department, other state agencies and the almost 15,000 Medicaid waiver providers. Most of the reports show summary and detail information on the status of claims being processed: processed claims; denied claims; paid claims. The data within the reports, especially the detail reports, lends itself to a spreadsheet type format. Each of the detail lines have the same variables displayed and in a spreadsheet format can be easily manipulated by the user to meet the needs of the user.

As SAS® has many built in procedures to easily produce Excel spreadsheets in a well presented format. Some of the ODoDD central office reports have taken advantage of this, but the vast majority of reports that are being produced are still text files for several reasons:

1. The Medicaid waiver providers are a very diverse group: from large agency providers with all the newest analytical software available to small independent providers that submit claims through public library computers and may have limited skills and access to computers. If all of the provider reports were produced in an Excel format, it would require the providers a way to consume the information. Text files are universally read by any device: from a PC, MAC, handheld device, phone.

   To address some of the needs of the expert users, ODoDD produces a fixed length file with no formatting for the majority of the claim detail files. These files can be read into a database or spreadsheet easily and can address some of the issues that large providers have.

2. The size of an .XLSx file is much larger than .TXT file with the same information. All the nice formatting of Excel spreadsheets come at a cost of additional disk space. Text reports can be designed to be easily read for the less expert users.

3. The resources, in both CPU and clock time, are much greater in producing Excel files versus .TXT files. The amount of time to produce the 30,000+ provider level reports in Excel would be tremendous.

The production of Excel spreadsheets has a place in ODoDD processing, but it is used strategically.

SILVER MEDAL

Multi-file Processing – SQL vs. SAS® MERGE file processing

In a ODoDD SAS® job there are many different methods of manipulating data from multiple datasets or tables:

- DATA/SET statement to combined one or more files into one with the same variables for all output data
- PROC APPEND procedure to add data from one or more files together.
- DATA/MERGE statement to join two or more datasets based on a common key to produce a common output file of the merged information.
- PROC SQL to use the power of SQL statements to combine data together in a more complex and typically more universally understood method.

There are many other ways to put together information from multiple locations, but the methods represent the overwhelming majority of the ODoDD file processing.

Each method has its advantages depending on what issue is being addressed. The difference between the DATA/MERGE and PROC SQL is a matter of complexity. A simple PROC SQL query can be written to accomplish the same goal as the MERGE with less code and more universal understanding: a SQL query is a SQL query. A programmer in SAS® can write a query that a .NET developer could easily comprehend. Similar queries can be used across multiple platforms. The issue can be with the performance of a query versus a more complex DATA/MERGE or DATA/SET method of multiple file processing.

Below is an example of code that was originally a multi-file processing DATA/SET that was rewritten as a PROC SQL statement:

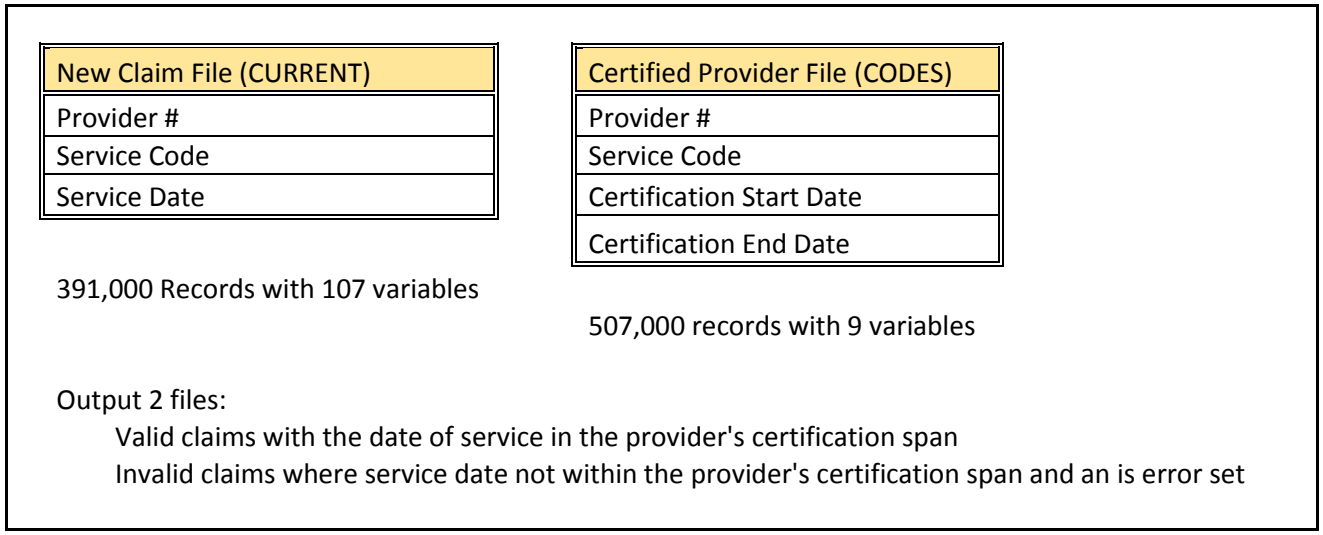


Figure 3 - SQL vs. SAS® Scenario

The code created using the DATA/SET statement is multi-file processing using LINK statements to control the processing flow:

```
 /*-------------------------------------------------------------------*/
 /* MATCH CLAIMS TO CONTRACTS AND DETERMINE IF SERVICE DATE FALLS     */
 /* WITHIN THE TIME SPAN OF THE CONTRACT.  OUTPUT CLAIMS WITHOUT A    */
 /* VALID CONTRACT TO A NOMATCH HOLDING FILE.                         */
 /*-------------------------------------------------------------------*/
PROC SORT DATA=CURRENT ;
  BY CNTRNUM SERVCODE SERVDATE ;

PROC SORT DATA=CODES ;
  BY FORM4NUM FORM4SC EFFCDATE EXPRDATE ;

DATA MATCH1 (DROP=CNTRTYPE) ERROR8 ;
  LENGTH CNTRNUM FORM4NUM $7 SERVCODE FORM4SC $3 ;
```

```
  IF _N_=1 THEN
  DO ;
    LINK GETFM4 ;
    LINK GETCLM ;
  END ;

CHKMATCH:  /* EVALUATES FORM 4-INPUT MATCH AGREEMENT */

  IF CNTRNUM=FORM4NUM THEN
  DO ;
    IF SERVCODE=FORM4SC THEN
    DO ;
      IF SERVDATE < EFFCDATE THEN
      DO ;
        ERR8= 8 ;
            OUTPUT ERROR8;
        IF NOT ENDCLMS THEN LINK GETCLM ;
          ELSE STOP ;
        GOTO CHKMATCH ;
      END ;
      ELSE IF (SERVDATE >= EFFCDATE) AND (SERVDATE <= EXPRDATE) THEN
      DO ;
         ERR8= 0 ;
        OUTPUT MATCH1 ;
        IF NOT ENDCLMS THEN LINK GETCLM ;
          ELSE STOP ;
        GOTO CHKMATCH ;
      END ;
      ELSE IF SERVDATE > EXPRDATE THEN
      DO ;
        IF NOT ENDFRM4 THEN LINK GETFM4 ;
          ELSE LINK ERROUT ;
        GOTO CHKMATCH ;
      END ;
    END ;
    ELSE IF SERVCODE > FORM4SC THEN
    DO ;
      IF NOT ENDFRM4 THEN LINK GETFM4 ;
        ELSE LINK ERROUT ;
      GOTO CHKMATCH ;
    END ;
    ELSE IF SERVCODE < FORM4SC THEN
    DO ;
      ERR8= 8 ;
       OUTPUT ERROR8;
      IF NOT ENDCLMS THEN LINK GETCLM ;
        ELSE STOP ;
      GOTO CHKMATCH ;
    END ;
  END ;
  ELSE IF CNTRNUM > FORM4NUM THEN
  DO ;
    IF NOT ENDFRM4 THEN LINK GETFM4 ;
      ELSE LINK ERROUT ;
    GOTO CHKMATCH ;
  END ;
  ELSE IF CNTRNUM < FORM4NUM THEN
  DO ;
    ERR8= 8 ;
     OUTPUT ERROR8;
    IF NOT ENDCLMS THEN LINK GETCLM ;
      ELSE STOP ;
    GOTO CHKMATCH ;
  END ;
  PUT 'EXECUTION ERROR: UNTRAPPED RECORD'  // _ALL_ ;
  RETURN ;

GETFM4:  /* INPUTS FORM 4 RECORD */

  SET CODES END=ENDFRM4 ;
```

```
  BY FORM4NUM FORM4SC EFFCDATE EXPRDATE ;
  RETURN ;

GETCLM:  /* INPUTS CURRENT CLAIMS */

  SET CURRENT END=ENDCLMS ;
  BY CNTRNUM SERVCODE SERVDATE ;
  RETURN ;

ERROUT:  /* HANDLES REMAINING CLAIMS AFTER END OF FORM 4 FILE */

   ERR8= 8 ;
   OUTPUT ERROR8;
  OUTERR :  LINK GETCLM ;
            IF ENDCLMS THEN STOP ;
            GOTO OUTERR ;
run;
```

The code written using SQL statements is:

```
 /*-------------------------------------------------------------------*/
 /* MATCH CLAIMS TO CONTRACTS AND DETERMINE IF SERVICE DATE FALLS     */
 /* WITHIN THE TIME SPAN OF THE CONTRACT.  OUTPUT CLAIMS WITHOUT A    */
 /* VALID CONTRACT TO A NOMATCH HOLDING FILE.                         */
 /*-------------------------------------------------------------------*/
PROC SQL;
 CREATE TABLE MATCH1_1 AS SELECT A.*,B.* ,(ERR8*0) AS ERR81
 FROM CURRENT A INNER JOIN CODES B ON A.CNTRNUM =B.FORM4NUM AND A.SERVCODE =B.FORM4SC
 WHERE B.EFFCDATE <=A.SERVDATE <=B.EXPRDATE ;
QUIT;

PROC SQL;
 CREATE TABLE MATCH1 AS SELECT DISTINCT * FROM MATCH1_1;
QUIT;

PROC SORT DATA=CURRENT ;
  BY CNTRNUM SERVCODE SERVDATE ;

PROC SORT DATA=MATCH1(DROP=ERR8 CNTRTYPE RENAME=(ERR81=ERR8));
  BY CNTRNUM SERVCODE SERVDATE;

DATA ERROR8;
 MERGE CURRENT(IN=A) MATCH1(IN=B);
 BY CNTRNUM SERVCODE SERVDATE;
IF A AND NOT B;
ERR8=8;

run;
```

The complexity of the DATA/SET with LINK statements is extremely complicated. The SQL version of the same much less complex and can be understood by a less SAS® knowledgeable programmer. The only issue is that the elapsed clock time for these two procedural steps using the same data on the same server and SAS® version:

DATA/SET with LINK	-	1.98 Seconds Clock Time	0.54 CPU Seconds
SQL	-	31.08 Seconds Clock Time	6.83 CPU Seconds

For this particular example, the amount of time is insignificant compared to the complex nature of the DATA/SET statement. Additionally, the DATA/SET method had issues with processing in some specific instances that produced erroneous results.

This performance, extrapolated to millions of records on each file indicates that the performance of SQL can be an issue if in this case there is a nearly 1500% increase in clock time and CPU time. There are SQL performance that could help in the time, but the overall performance of the two methods is the reason why the multi-file processing using SQL gets the Silver Medal.

GOLD MEDAL

Agility of MBS Code with the Use of SAS®

The mission statement for ODoDD is:

> The Ohio Department of Developmental Disabilities (DODD) oversees a statewide system of supports and services for people with developmental disabilities and their families. DODD does this by developing services that ensure an individual's health and safety, encourage participation in the community, increase opportunities for meaningful employment, and provide residential services and support from early childhood through adulthood.
>
> The mission of DODD is continuous improvement of the quality of life for Ohio's citizens with developmental disabilities and their families. Our vision is that Ohio's citizens with developmental disabilities and their families will experience lifestyles that provide opportunities for personal security, physical and emotional well-being, full community participation, productivity, and equal rights.

In order to facilitate this mission, the providers of the Medicaid associated services need to be compensated. The MBS system is written to:

- Validate the claim in terms of service, individual being served and the provider of service.
- Ensure that the costs of the claim stay within the limits set.
- Send a file to our sister state Medicaid agency; which approves or denies the claim based on the Medicaid eligibility of the provider and individual.
- Payment for claims that are approved by our sister state Medicaid agency is sent to the provider.
- Provide claim information to the backend reporting systems.

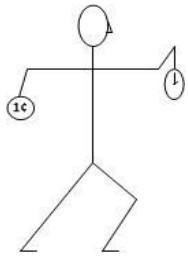
Along with the standard weekly processing, the MBS system:

- Be able to react quickly to issues.
- Be able to support changes as the approach to the department's mission changes.
- Provide Ad Hoc reporting for Central Office.

The use of SAS, its associated procedures, statements and file structures, is a vital component in making the MBS system successful. On a weekly basis, the approval rate of the sister state Medicaid agency adjudicated claims is about 99.7%. This is due to the vigorous scrubbing of the claims before sending the file to the sister state Medicaid agency for adjudication.

In exception conditions, the use of SAS® has been a vital component in getting to an interim resolution before a final solution can be found. The flexibility of code and the ability to use both SAS® files and SQL databases has been critical.

THIRD EVENT – SAVINGS



Savings

- Processes and procedures to save time in development.
- Efficient utilization of available tools and equipment.
- Proper utilization of the skills of the limited staff

ISSUE TO RESOLVE

As with the Flexibility event, the Savings event does not address one specific issue. The contenders of this event were evaluated based on the savings they achieved in delivery time to production, resource requirements, disc and system resources and actual funds. Each of the winners and losers will have the issue that it was attempting to address and the success of the approach.

Some of the potential resolutions that were tried that did not make the podium include:

1. Issue: Processing files after using file compression products

   To save on transmission time, many of the files that are transmitted between the sister state Medicaid agency and ODoDD are compressed. This saves tremendously on space and on transmission time, but using the files after takes manual intervention and the process can get cumbersome. The use of this in our specific environment is best used in permanent backup file storage and transmission of very large files.

2. Issue: Job Scheduling and SAS® Enterprise

   When evaluating the SAS® environment when ODoDD was moving from the VAX to the Server, the SAS® Enterprise package was evaluated. The advantages of Enterprise were evident from the creation of code and jobs, but the additional cost of the Enterprise license was prohibitive in relation to the size of the system it was to support. In addition, there was a scheduler in place. This jobs scheduler can be used to submit the SAS® programs similarly to SAS® Enterprise.

   This decision was more cost effective and saved training costs for the production control staff, but it lost the advantages that could have been gained through SAS® Enterprise.

BRONZE MEDAL

Using SAS® Compress= Data Set Option to Reduce Disk Usage

The Data Set Option of COMPRESS= reduces the number of bytes that are required to represent each observation when writing a file. Advantages of compressing a file include reduced storage requirements for the file and fewer I/O operations to read or write to the data during processing. However, more CPU resources are required to read a compressed file (because of the overhead of uncompressing each observation). Below is a quick comparison of the cost for a file that had 19,549,538 records running on a Windows Server 2012 R2 using SAS® Version 9.4.

Function	Uncompressed file		Compressed File		Difference	
	Clock Time (in seconds)	CPU (in seconds)	Clock Time (in seconds)	CPU (in seconds)	Clock Time	CPU
DATA/SET to compressed file	58.6	35.45				
DATA/SET to a temporary file	39.2	3.93	50.75	19.43	29.46%	394.40%
DATA/SET to permanent file	37.85	4.2	51.12	19.37	35.06%	361.19%
PROC SORT to temporary file	54.07	28.89	61.78	42.18	14.26%	46.00%
PROC SUMMARY	10.95	5.03	23.54	19.75	114.98%	292.64%

Table 6 - Processing a Compressed and an Uncompressed File

The table shows there is some clock time degradation by processing a compressed file, but not significant. The amount of space that was saved by compressing this file was 28.41% of disk space and reduced the file size from 8.4mil KB to 5.9mil KB. This file has not contain a lot of white space per record.

The reasons why this is only a Bronze Medal winner is:

- Compression works better on some files and not as well on others.
- In an environment where storage is not at a premium, the cost in CPU and Clock time may not be worth it.
- This is best used for files that are processed on a limited basis, so in an archived or backup environment it is ideal.

SILVER MEDAL

“Plus-ing”…or reviewing code surrounding new change

A piece of SAS® code can be one or two simple procedures to produce a report or thousands of more complex procedures strung together in a time critical application. The more complex and older the code becomes, the more sections of the code become obsolete as new code is added to address different issues.

The term “plus-ing” is one a manager of mine used many years ago to do a quick evaluation of the code surrounding the place where new code is being placed to determine if it is still relevant. If the effort to delete old comments or remove the processing that no longer in use does not alter the deliverable by more than 10%, then invest the time in removing and add testing to ensure that there are no processing differences.

This approach does not mean to redesign an entire piece of code when the task is add a couple of new lines. It is more intended to make the code more easily read and understood. The approach is also to remove exception coding that is no longer relevant.

The effort will already be there to code, unit and system test and implement are already in the change lifecycle. The removal can make code easier to read, especially by those not familiar with the specific functionality the old code addressed. If the effort to remove the changes significantly alter the end deliverable, then the task needs to be a project unto itself.

By removing the dead code and comments, it helps everyone that tries to make updates in the same place in the future. In some cases, contractors are brought for specific large projects. By plus-ing, it makes it much easier for new people to come up to speed.

In the ODoDD case, there are pieces of code where the functionality has not change in the 15 to 20 years it has been running. There are new services, providers and individuals, but the basic processing has not really changed a lot in some of the areas. So that when I come across a service that is no longer supported, either through a new claim or an adjustment, I tend to try to remove the references to it. Or when a comment says that it was an emergency fix from 6 years ago, I remove the comment and leave the code because it is just standard code now.

The reason this is not a gold medal winner, is that the effort has no significant affects to processing or true dollar cost savings at the time it is done. It aids in the legibility of the code and will help future development by removing code and comments that are no longer active.

GOLD MEDAL

Periodic review of the notes and warnings on a SAS® log files

The SAS® log results from the execution of a SAS® program including:

- Name and number of observations of each file processed.
- Number of variables and can include more detailed information about files being processed.
- CPU and clock time for SAS® program execution
- Notes which are informational message which does not stop processing but can indicate that the program is not programmatically correct.
- Warning messages which alerts of a potential issue, but does not stop the execution of the program.
- Error messages which indicate a significant issue with the code and either stops processing or flags the error and continues processing.

When testing a program, the log is a vital tool in determining the success of a SAS® program. The warnings and errors are reviewed. Warning messages are reviewed so that decisions can be made to allow known exceptions to go into production or a fix to the code or data to eliminate the warning messages. Error messages are reviewed and fixed.

Once the code is implemented into production, the logs are typically reviewed if there is an issue like: increases in clock time, CPU or disk usage; production support issue research; or analysis when implementing new code. ODoDD has found it extremely beneficial to periodically review the logs.

With the implementation of additional code and data over time, new warnings and pertinent notes can begin to appear. As neither a note nor warning will stop SAS® programming from processing, they could indicate the code is no longer functioning as originally designed. There may be data issues further upstream that could affect the processing that, although not an issue that is critical enough to stop processing, could negatively affect the end results.

The periodic review is a great method to identify issues in a more timely way than waiting for a user to discovering them. The accuracy of results is vital and to find issues before they arise assures it. Therefore, the periodic review receives the gold medal in this event.

CONCLUSION

Performance, flexibility and cost savings are critical to running any IT shop, but especially a small one. The Gold Medal winning solutions all had a common thread of staying current and in front of issues. Keeping the system and code fresh and green is vital to ensure success. There is not always time to be thinking ahead but when there is an opportunity either when there is some time, think to what can be done to make the code better.

REFERENCES

SAS(R) 9.4 Companion for Windows, Fourth Edition

ACKNOWLEDGMENTS

Randall Blackstone, Ohio Department of Developmental Disabilities, Software Development Specialist 4

Jaimey Karhoff, Ohio Department of Developmental Disabilities, Medicaid HAS 2

RECOMMENDED READING

- *Base SAS® Procedures Guide*
- *SAS(R) 9.4 Logging: Configuration and Programming Reference, Second Edition*

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Stephen C DeVoyd
Ohio Department of Developmental Disabilities
(614) 728-0514
Steve.DeVoyd@dodd.ohio.gov
DODD.Ohio.gov