Computer Science and Information Technology 7(2): 31-39, 2019 http://www.hrpub.org
DOI: 10.13189/csit.2019.070201

GDG in UNIX? No Way!

Kannan Deivasigamani

Business Technical Analyst - SAS Team, Wellcare Health Plans, Inc., Florida, USA

Abstract IBM mainframes in the z/OS environment provide a generational structure often referred to as Generation Data Group (GDG) for file storage to maintain data snapshots of related data.[1] These data resulting from business operations within a servicing organization are not uncommon. This structure can hold TEXT data sets without a problem. However, in the case of a UNIX or Linux platform, a comparable structure is unavailable for use by SAS for storing data as TEXT files. This paper contains a solution to this problem and shows a comparison of what the mainframe GDG offers and the solution offered. A developer or a programmer may find that the solution, TextGDS (SAS macro) is even better than the mainframe GDG structure in certain respects. Although there are both limitations and delimitations when using TextGDS, the tool helps to fill the void with UNIX-SAS.

Keywords TextGDS, SAS, GDG, MAINFRAME, UNIX-SAS, LINUX-SAS

1. Introduction

Developers around the world who work in a mainframe environment commonly use GDGs for storing the different types of hierarchical data required to support their businesses. The feature is available in mainframe as part of the operating system with IBM OS 3090, Z/OS, and others, where job control language is commonly used to embed SAS statements as a program to run in a batch-mode. While JCL allocates resources on the mainframe on Linux and UNIX platforms, it is managed by the operating system and the infrastructure accessed by the batch programs.[2] It is also a normal practice to have GDGs defined to store different types of data depending on the needs of the business for a specific number of generations. S*AS* offers a generational data set structure as part of the language feature that many users are familiar with, use in their organizations, and manage using keywords such as *genmax* and *gennum*. Although SAS operates in a mainframe environment, users also have the ability to tap into the GDG feature available on Z/OS and on other applicable mainframe platforms.

1.1. Need to Migrate from Mainframe System

The mainframe end user computing cost is increasing and has demanding business intelligence and operational reporting on structured data. Even though mainframe is billed by usage, optimization is an essential part of the information technology (IT) group and leads to expensive bills based on usage of Logical Partitions (LPAR) [3]. With cost-saving initiatives across businesses, and due to some scaling factors, many organizations are in the process of migrating from mid-tier platforms to cheaper operating platforms such as UNIX and AIX. With Linux being open source and a cheaper alternative, several organizations have opted for the UNIX distribution of SAS that could work in UNIX/AIX environments. Metaware, an IT organization, migrated 12,000 million instructions per second (MIPS), saved over $55 million a year, and recouped the cost of conversion in 12 months [4]. MIPS is a common industry term that refers to million instructions per second and a measure of computing performance and implies the amount of work a larger computer can do and is a way to measure the cost of computing. Organizational leadership encourages employees to optimize their processes to save on end-user computing costs.[5]

Two forms of cost savings in organizations are (1) measures taken to downsize human resources and (2) migration from mainframe to less expensive systems such as UNIX or LINUX operating systems.[6] Based on the research, several organizations are moving away from mainframe and could benefit from the tool discussed here, depending on their destination platform.

1.2. Research on State-of-the-art migration Projects

A detailed literature review reveals that several organizations have attempted migration from legacy systems such as mainframe to a client-server or a cloud based servicing environment. An insurance giant Prudential used a third party, Cedar knowledge solutions for their data migration[11]. An Extract transform load (ETL) tool named Sagent to pull out data from 15 core

systems into a data staging area and then moved into a data warehouse environment for everyone else to pull data from[11]. Security Industry Automation corporation hired Clerity Solutions Inc to help with their Mainframe to Unix migration; Readers Digest Association outsourced to Infocrossing; SIAC a subsidiary of New York Stock exchange Group Inc migrated to IBM servers running AIX. [12] Taking the approach of mapping exercise to map data from variable to variable across all files and then moving them based on a structure is a time consuming and expensive. None of these research articles have mentioned in detail to the granularity of the data type and file type to be able to compare against what they did with mainframe GDG files. Some organizations based on the authors research show that they couldn't migrate due to the sheer size and complexity of the systems they had on mainframe.

1.3. Challenges

While migrating to UNIX or Linux may be a viable alternative, the migration effort brings up certain nuances to technical conversion teams. A migration effort from mainframe to UNIX or LINUX operating systems is not easy; rather, it is time consuming and involves complex data patterns and mapping.[7] Unlike mainframes, the concept of GDG does not exist on UNIX. Although SAS offers a generational data sets feature as part of the language, the feature is only good for SAS data sets. If an organization needs to house and operate with a GDG-like structure for TEXT data sets, there is not one available in a UNIX/AIX environment. Storing non-SAS types of data in a UNIX/AIX environment becomes a challenge and makes the conversion process more challenging. Financial organizations operating mortgage lending businesses require data sets related to subprime mortgage analytics, incentives, and regulatory reports to be shifted to UNIX/AIX environments. Many of these files are generational, and the paucity of literature and research on this topic is evident. This paper contains information that reveals the basics of a mainframe GDG and indicates how the TextGDS tool is likely to help with such a migration effort.

The development of the macro was not an easy task and took much effort to identify the challenges and then arrive at solutions to those challenges. It was a step by step effort to arrive at the final end-product that would work. Several design challenges were encountered both in the design stages and the development stages of the macro. The author had to parse the file name into parameters that could be passed on from a SAS program to the UNIX operating system as a file name. There was meta-data information about the existing GDS files required before processing the call. The information about how many files exist currently in the location, the oldest generation number, highest generation number we also required. The author had to develop SQL to obtain intelligence so a new generation number can be assigned to the new file. In addition, the rule also necessitates that there can only be a specific number of active generations available which is a characteristic of the dataset. This value is also passed on as a parameter during the macro call by the calling program. Internally, this means that the macro should be able to delete the oldest file if the max capacity is reached. The author had to research to find the code that would delete a file from UNIX or LINUX operating system and invoke that step from within the SAS macro. These were some of the main challenges while trying to keep things simple to the user, the author has also tried to encapsulate everything within the macro to maintain some level of abstraction and the user just needs to know how to call the macro and the macro will get the job done.

2. Mainframe GDG

IBM has built a cataloguing mechanism for successive file updates or generations of related data that are referred to as GDGs.[8] Each data set within a group is called a generational data set (GDS). These generational data sets are related to each other in a chronological order. If the attributes of these data sets are identical, they can be retrieved together within a batch process if necessary. They can also be retrieved individually or by a relative reference as the most recent generation, the oldest generation, or any of the *n*th generation in between, which is counted backward from the most recent as the latest and the lowest nonnegative number, which is a "0".The maximum number of generations that a GDG can hold is 255. While executing on a mainframe environment, SAS can access the GDG feature, write SAS data into the generational datasets, and text data into the generations of a GDG. When these files are migrated to the UNIX/AIX environment, the structure needs to be available in the target platform. This is where the TextGDS tool becomes useful.

2.1. What is TextGDS?

TextGDS is a SAS macro that accepts specific parameters and populates the macro variable &fn that will hold the appropriate file name. This macro variable "&fn" will be used by the calling program within a file name statement as shown below and the file will be part of the Text Generational Data Set.

```
filename f1 &fn;
```

These specific parameters have some rules that are described in the "Macro Rules" paragraph below. This macro was designed by trial and error experimentation and by finding out scenarios that were handled one after another and was gradually built to reach a stable version that can handle many of the common scenarios.

The author, while developing encountered issues during run time tests and also issues that were encountered by

other users during an initial beta release that helped bring to surface, scenarios that were to be handled. With further development, the author was able to fix those and ultimately reach production standards. A SAS program in general will be considered "ready" for production, if tested successfully using variety of data with production scenarios that the program is expected to handle in a specific time without errors without any performance issues. The results are then validated by developer, tester, subject matter experts, and business users and finally signed off for deployment to production. After a series of such tests, the macro was promoted to production and used by several production process by a banking organization for archiving all of the production files. It is unfortunate that due to business divestiture decisions by leadership was sold to several other companies and that specific line of business no longer exists today with that multi-national corporation. While the business was sold, the macro TextGDS is still available for use to anyone who needs it.

2.2. Macro Rules

The program code under the section labeled "MACRO CODE" contains the actual SAS macro that a user would have to include to be able to call the macro. The macro needs to be called with the following syntax %TextGDS (f1=, f2=, n=, m=) where f1 should point to a directory structure or path on UNIX where the GDS files can be stored. The user or process needs to have "write" access to the folder. The second parm f2 should point to the file name of the GDS. The third parm 'n' should point to the generation prefixed by a sign. The 'n' value cannot be without a sign and the macro will not operate as expected. The macro variable accepting values without sign can be coded to can be part of future enhancement. The last parm m has to be a numeric value without a sign and have a value greater than 0. Having a negative or a sign value here can lead to undesired results. This can also be a future enhancement to accept input values prefixed with positive sign.

3. TextGDS Macro Calls

TextGDS can be called by SAS programs that need a structure similar to a mainframe GDG for storing TEXT data in an AIX/UNIX environment.[9] The macro will require some parameters to be passed that will determine the kind of operation a user intends to perform. The operation can be better explained with the example provided below that uses the macro *TextGDS* explained throughout this document whose code is included at the end. The macro call for this example is as follows:

%TextGDS (f1=/data, f2=Text_, n=+1, m=5);

Such a call would result in populating a macro variable *&fn* with a value that will be assigned to the *file-ref* name in the *filename* statement. This value assigned to the macro variable *&fn* will be named in such a way that it resembles a GDG structure. The file created with this &fn will be part of the generational dataset as per the macro call by the calling program. The file can be read in an *SAS* program using the same *SAS* macro by altering the parameters (parms) to fit a user's need.

An example of a read is demonstrated below. In the code below, the parameter *"n"* has a value of *"+0,"* which indicates that the user is requesting the macro to point to the current generation of the text file.

```
%TextGDS (f1=/data, f2=Text_, n=+0, m=5);
Filename f1 &fn;
```

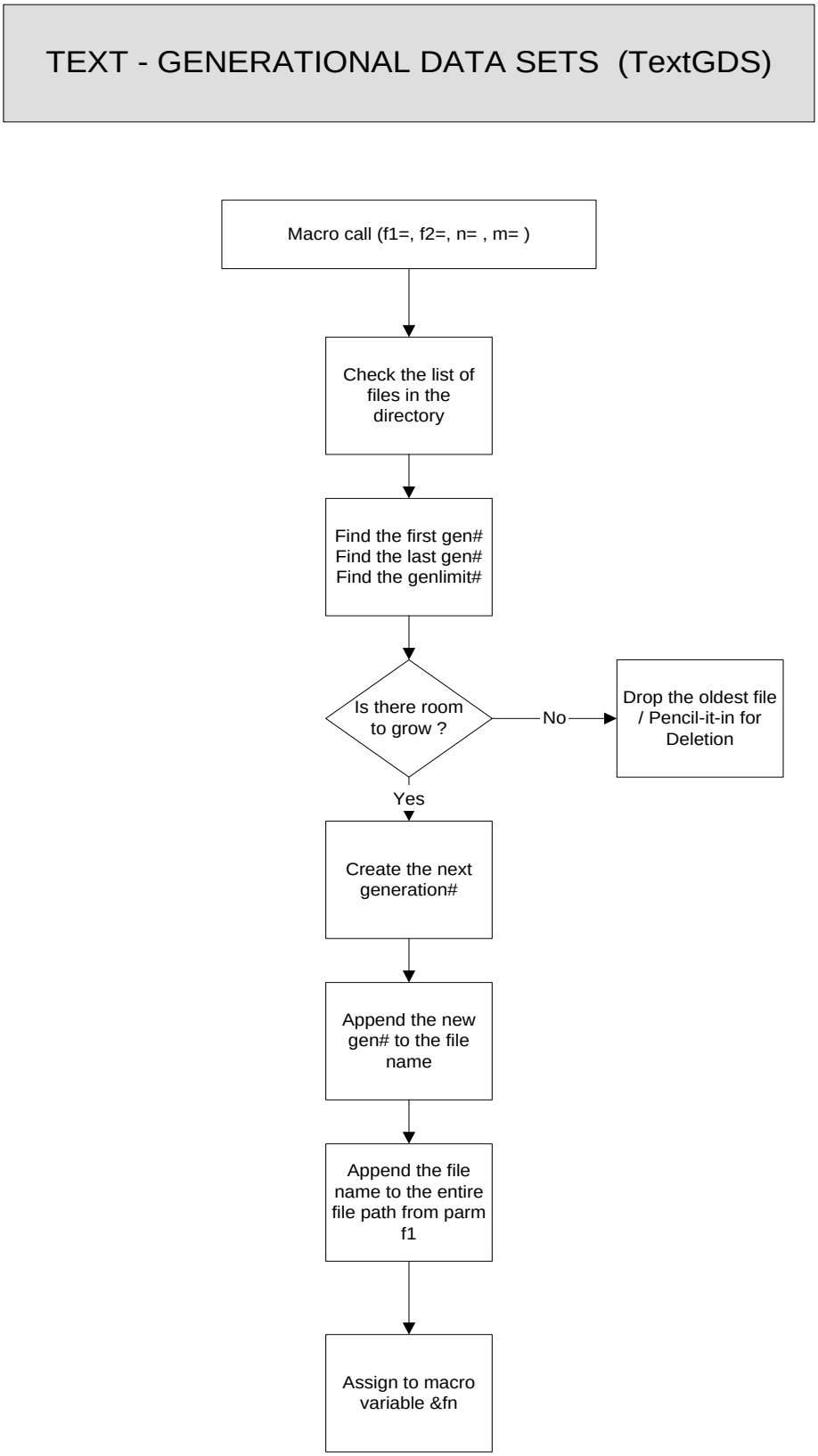

Figure 1. A logical operational view of TextGDS

4. Macro Operation

Parms *f1* and *f2* point to file location and file name, respectively. The next parameter n can have either a positive or a negative value. It cannot have an unsigned numeric value. The last parameter, which is *m* in this case, holds a value of 5, which indicates that this macro is set to hold a maximum of five generations. This is equivalent to the *GENLIMIT* parm on the mainframes while defining a GDG.[10] If the file in the directory is empty before running this macro and is run for the first time with parms shown below, the first generation will be created in accordance with the macro parameters provided. Figure 1 describes the operational flow of the TextGDS macro.

```
%TextGDS (f1=/data, f2=Text_, n=+1, m=5);
Filename f1 &fn;
Data _null_;
        File f1;
        Put 'the TDS# is:' &fn;
Run;
```

The data folder will have the following file:
/data/Text_000001m005.dat

As shown above, the first part is from parm ***f2=Text_*** and the second part is from parm ***n=+1***, which resulted in **000001.** The last part is from the parm ***m=5*** that resulted in 005, which is appended with the file extension ".dat" that completes the file name and results in

Text_000001m005.dat

If the code segment shown above with the macro is invoked again, then the next generation is created and the ls –altr command in the /data folder will show the following two files:

Text_000001m005.dat
Text_000002m005.dat

If the code segment is executed again, the third generation will be created and will have three files as shown below:

Text_000001m005.dat
Text_000002m005.dat
Text_000003m005.dat

A repeat of the same will result in the fourth file and finally another repetition will result in five files as shown below:

Text_000001m005.dat
Text_000002m005.dat
Text_000003m005.dat
Text_000004m005.dat
Text_000005m005.dat

And if the code is executed again, the sixth generation will be created and most recent five generations will be available for use as shown below:

Text_000002m005.dat
Text_000003m005.dat
Text_000004m005.dat
Text_000005m005.dat
Text_000006m005.dat

The oldest generation ***#000001*** will be deleted and the most recent five generations will be available. The oldest generations will keep dropping off as new generations are created. The macro internally manages these generation retention processes using the work space mentioned earlier on in the paper. The macro reads the metadata in the current directory which is nothing but the list of generations matching the file name provided in the macro parameter and stores it for file management operations in the macro. The maximum and minimum generations are identified, and their generation numbers are handled depending on the max gen limit that the macro initially defined it to hold. If the user is requesting a +1, then the generation number is incremented to the next higher number and the oldest generation is deleted within the macro code. So far, we have seen examples of creating new TextGDS files but not reading in existing current and historical generations which is discussed in the next section.

5. Macro Read Operation

A macro call with a negative sign to parameter "*n*" will result in a read operation of a historic generation as specified by the numeric value assigned to the parameter "***n***".

```
%TextGDS (f1=/data, f2=Text_, n=-1, m=5);
Filename f1 &fn;
Data _null_;
   File f1;
   Put 'the GDS# is:' &fn;
Run;
```

A macro call with a value of ***-1*** will result in reading the previous generation of the file. As an example if we have the following five files in the /data folder if a read is attempted with a ***-1*** as described, ***Text_000005m005.dat*** will be referenced by the ***&fn*** macro variable and assigned in the ***filename*** statement.

Text_000002m005.dat
Text_000003m005.dat
Text_000004m005.dat
Text_000005m005.dat
Text_000006m005.dat

Similarly, if a -2 is the value to the parameter m, then ***Text_000004m005.dat*** will be the file name that the ***&fn*** macro variable will be pointing to as a result. If the macro is referencing a generation that is out of scope, then as one would expect, the code would not be successful in pointing to the generation.

The macro is intelligent enough to recognize what the user is requesting, to perform the operation in the UNIX

region using a temporary work space for internal calculation, and to populate the final output variable, which is nothing but a path, followed by the file name that includes the fixed name along with the variable generation number followed by an extension indicating the generation limit for which the file structure was defined. This is the short version of the macro operation. A detailed explanation is provided with the example discussed in this presentation.

6. Limitations and Delimitations

Table 1 shows the similarities between mainframe GDG files and TextGDS files. They are very similar and can be used without much complexity. In many cases, the TextGDS feature offers more flexibility than what is currently available on the mainframe platform. There are certain limitations to the current version of the macro. For example, the maximum value that can be held by the three-digit gen-limit value is 999, which is identical to a SAS GDS. In addition, the highest generation number of the TextGDS indicated by the six-digit value that follows the TGDS name is 999,999 which is about a million generations. The generations may recycle, but the current version of the macro is not capable of recognizing the smaller number after recycling beyond 1,000,000. Upon reaching the limit, a new name can be defined and can start another set of million files, which would be a simple alternative to reaching the max count. Mainframe GDGs require a backup, if required to retain more than 255 generations while the current version of TextGDS allows a much larger retention count before necessitating a backup.

The number of generations in TextGDS can also be adjusted by shrinking or expanding the six digits to fit one's needs. The maximum number of generations may be shrunk or expanded according to one's needs. The extension of the file, which is ".dat," may also be modified to ".txt" or another type depending on the need of a project. All three limitations discussed may also be converted to parms that can be input to the macro to accept the values dynamically as the macro is invoked. However, all these enhancements call for a code change in the macro and in the way the macro will be invoked.

Another limitation or requirement is that this macro requires temporary work storage for the computation where the meta file is written and cleared out at the end of the macro execution, leaving no trace. The work space or the temporary folder is a requirement in the current design. One can choose to have this provided in the parm if desired; however, in the current setup, the invoking job will have a subfolder within the work folder and will be used for the computation. Depending on the infrastructure, program, and macro setup, the user might choose to add a command "mkdir tmp" in place of the work folder as appropriate.

The macro does not prevent anyone from pointing to a "+5" or "> +1 " without warning the user of getting ahead of the gen numbers. Therefore, the macro needs to be used with caution, and it will be the responsibility of the user to invoke it with appropriate parameters. Checks in code may be included to look for out-of-bounds values while invoking the macro to ensure appropriate functionality. In addition to the above, the macro currently expects a sign while referring to the current generation which was explained in the paragraph above named “Macro Rules”.

Table 1. Mainframe GDG vs. TextGDS – a comparison

Mainframe GDG/GDS	TextGDS
Holds maximum of 255 generations	Hold 1,000 but customizable to hold more
Refer using relative or absolute reference	Can be referred using relative or absolute reference
Built-in within the O/S	Macro call is required with the right parameters
“+1” will create a new generation	Macro call with “+1” is required as a parameter
Can call all generations at once	Need to specify each generation to invoke all files together
Can hold any type of data	Can hold any type of data but specially designed for non-SAS data
For mainframe only	For UNIX/AIX only. Using this concept in other platforms may require modifications to the macro and possible scripting depending on the platform
Mainframe has no file extensions	Currently has TXT but can be customized to have another extension such as .BIN, .TXT, etc.

7. TextGDS - Macro Code

```
%macro TextGDS(f1=,f2=,n=,m=);
  %sysexec %str(cd &f1; ls &f2* > &WORKFLDR/meta.txt);
  filename f "&WORKFLDR/meta.txt";
  data in;
       infile f length=recl;
       input @1 inrec $varying100. recl;
       newin = trim(inrec);
       dat_pos = index(inrec,".dat");
       len = length(newin);
       nc = substr(newin,dat_pos-10,6);
       n  = input(nc,6.);
       mc = substr(newin,dat_pos-3,3);
       glimit = input (mc,3.);
       put nc n mc glimit;
  run;

  proc sql;
       select  count(*)             ,
               min(n)               ,
               max(n)               ,
               glimit
         into :calc_tot             ,
              :calc_min             ,
              :calc_max             ,
              :calc_glimit
         from in;
  quit;

  %let abs_n = %sysfunc(abs(&n));
  %put ' &abs_n &calc_tot &calc_min &calc_max &calc_glimit:'
       &abs_n &calc_tot &calc_min &calc_max &calc_glimit;


  %if &calc_tot ne 0
  %then
    %do;
       %put '==if part' &calc_tot ;

       proc sql;
            select max(n) &n format=z6.
              into :newmaxn
              from in;
       quit;

       %put '&maxn=' &newmaxn;

       proc sql;
            select  min(n)       format=z6.,
                    glimit       format=z3.
              into :dropgen,
                   :gen
              from in;
       quit;

       %put '&newmaxn &dropgen &gen=' &newmaxn &dropgen &gen;
       %let igen  = &n+0;
```

```
        %if &igen ge 1
        %then
          %do;
              %put '**inside igen**';
              %let drop_file_name = &f1/&f2&dropgen.m&gen..dat;
              filename dropfile "&drop_file_name";

              data _null_;
                   %let calc_newhi = &calc_max + &n;
                   %put '==&calc_max &n &calc_newhi:' &calc_max &n &calc_newhi;
                   rc=fdelete("f");
                   %put '&calc_glimit &calc_tot &m='  &calc_max &n &calc_newhi &m;
                   %put 'sysevalf:' %sysevalf(&calc_glimit+1);
                   if (%sysevalf(&calc_glimit+1) le &calc_tot) and (&n gt 0)
                   then
                      rc=fdelete("dropfile");
              run;
          %end;
        data _null_;
              %global fn;
              %let fntemp = %qsysfunc(dequote(&f1))/&f2&newmaxn.m&gen..dat;
              %let fn = "&fntemp";
        run;
        %put '==&fn:' &fn;
    %end;
  %else

   %else
     %do;
         %put '==else part';
         data in2;
              m=&m;
              m2=put(m,z3.);
              infile f length=recl;
              input @1 inrec $varying100. recl;
              newin = trim(inrec);
              dat_pos = index(inrec,".dat");
              len = length(newin);
              nc = substr(newin,dat_pos-10,6);
              n  = input(nc,6.);
         run;
         data _null_;
              mchar  = &m;
              glimitc=put(mchar,z3.);
              put '==glimit mchar:' glimitc mchar;
              call symput('glimit',glimitc);
         run;
         data _null_;
              %global fn;
              %put '==&m:' &m;
              %if &n gt 0 %then %let fn="&f1/&f2.000001m&glimit..dat";
         run;
     %end;

   %if (&calc_tot lt &abs_n) and &n lt 1 %then %let fn='FILE NOT FOUND ERROR!';

%mend TextGDS;
```

```
%let WORKFLDR=%sysget(BASE_WORK_PATH);
```

8. Explanation of Macro Code

The code above is a SAS MACRO named TextGDS and it accepts the 4 parameters f1, f2, n, and m. The first part of the code looks at the path provided in the first parameter f1 and finds out how many generations of the specified file name already exists in the location. With this information, the code knows the next generation number, if it has reached the max generation, and what the next generation number should be to create a new generation of text file. If the max number of generations has reached, it also knows which generation needs to be dropped from the list to maintain the count, if a new generation is called to be created through the parm that is passed to the macro.

The next part of the code drops or deletes the oldest file to maintain the maximum number of generations when a "new generation" (+1) is requested through the macro call. Based on the calculations in the first part of the code, the new generation number is calculated and assigned to the output variable (&fn). The variable becomes the file assigned to the "FILENAME" statement in the calling SAS program. The *.meta data file is temporary but can be reused once the contents are cleared or a new file is created overwriting the contents of the existing file. It is advisable that the temp folder is associated with the program name so concurrent or parallel use of the workspace does not have interference during multi-threading and provides an independent space for each program.

9. Conclusions

TextGDS is a very easy tool to maintain and use and can be considered as another SAS program. The tool can be enhanced to make it more robust but it is a good start and can be used by developers. The TextGDS feature may someday become available as a part of native SAS language for developers to use to maintain generational text files when organizations need them. Although enhancements are possible, despite the existence of both limitations and delimitations, the TextGDS tool may be useful for software developers working with dual platforms who seek to use the solution. Organizations with initiatives of cost savings with technology can benefit from this tool, depending on their choice of target platform.

10. Recommended Reading

- Base SAS® Programming Guide
- SAS® Macros Programming Guide
- Linux/UNIX

REFERENCES

[1] SimoTime Technologies and Services. Generation data group [Internet]; 1987. Available from: http://www.simotime.com/gdgone01.htm.

[2] Volovik V, Edwards E. Migration from Mainframe to LINUX: Leading but Still Bleeding Edge [Internet]; 2009 [cited 2018 Sep 29]. Available from: https://www.lexjansen.com/nesug/nesug09/ap/AP02.pdf

[3] LeRoy M. Why is my mainframe system so expensive? [Internet]; 2018 [cited 2018 Sep 28]. Available from: https://www.zcostmanagement.com/why-is-my-mainframe-system-expensive

[4] Metaware. 12,000 mainframe MIPS moved to Unix. n.d. [cited 2018 Sep 29]. Available from: http://www.metaware.fr/pdf/1_Metaware_French_Social_Security_Mainframes%20Migrations_Case_Study.pdf

[5] Deivasigamani K. Relationship between leadership and mortgage banking end-user computing efficiency [dissertation]. Ann Arbor: University of Phoenix; 2016. 136 p.

[6] Beheshti HM, Bures AL. Information technology's critical role in corporate downsizing. Ind. Manage. Data Syst. 2000;100(1):31-5.

[7] Syncsort. Considerations for Mainframe Application Modernisation [Internet]; 2010 [cited 2018 Sep 29]. Available from: https://whitepapers.em360tech.com/wp-content/files_mf/white_paper/mod60wd_appmodernization_u_app2_0.pdf

[8] IBM Knowledge Center. What is a generation data group; [n.d.] [cited 2018 Sep 16]. Available from: https://www.ibm.com/support/knowledgecenter/zosbasics/com.ibm.zos.zconcepts/zconcepts_175.htm

[9] Deivasigamani, K.Text generational datasets [TextGDS]; 2017 [cited 2018 Sep 10]. Available from: http://support.sas.com/resources/papers/proceedings17/0274-2017.pdf

[10] SAS Institute. Generation Data Sets [Internet]; 1999 [cited 2018 Sep 29]. Available from: https://v8doc.sas.com/sashtml/lrcon/z0934566.htm

[11] Howard LS. Extreme makeover: Migrating data. National Underwriter 2003 Jul 28;107(30):19.

[12] Thibodeau P. Mainframe Migrations Follow Different Routes. Computerworld 2007 Jan 15;41(3):14.