Paper 295-2013

CLISTS: Improve Efficiency of TSO Applications Using Mainframe SAS®

Dr. Russell Jay Hendel, Towson University and CMS

ABSTRACT

Have you been spending a few hours every month submitting several dozen SAS® jobs to mainframe systems using an IBM TSO environment with the Interactive System Productive Facility (ISPF)? You know that within SAS, SAS macros can efficiently manage repetitive tasks; but how do you manage repetitive tasks with JCL, the TSO control language? CLIST is precisely what you need: It enables you to automate repetitive tasks that use JCL and SAS. CLIST is an easy language to learn, requiring no former knowledge and using only a handful of basic commands. We present illustrative CLIST code covering basic groups of CLIST commands. People already familiar with JCL and SAS who write jobs using both of them will benefit from this presentation.

INTRODUCTION and GOALS

This paper describes the CLIST programming language. The CLIST language may be used within the TSO environment on an IBM mainframe machine. In fact, CLISTS are written and stored in the same way that JCL code is written and stored. Consequently, this paper assumes familiarity with TSO, JCL and mainframe SAS.

Why would you want to use a CLIST? The CLIST can routinize complex sets of mainframe tasks saving the programmer time and reducing errors. The entire CLIST language with illustrative programming examples may be found in the CLIST manual. This paper takes a task of intermediate complexity, producing data for a triangle payment report, to illustrate key features of the CLIST language including, interaction with the user, modularization, basic arithmetic and string functions, special useful CLIST functions, as well as writing and submitting, via the CLIST, TSO and SAS code. Upon reading this paper a reader will have the ability to routinize groups of related mainframe tasks using a CLIST.

TRIANGLE REPORTS

Triangle payment reports occur in a variety of settings. A sample triangle report with explanatory comments preceding it, is presented in Table 1. The actual example inspiring this paper produced data for a three-year , 36 x 36 triangle report. When I was assigned the task, I was surprised to find that one programmer routinely downloaded 36 files from the mainframe into an ACCESS database which then extracted key data elements to produce the report.

However, this is unnecessary and wasteful of time and resources. For example, one does not need each of the entire 36 monthly payment files; rather, one modestly needs a summary of payments and the months they were allocated to. In other words, a simply *proc means* summary is all that is needed from each of the 36 files. The 36 proc summaries can then be coalesced as the final mainframe output. The actual triangle report can then be produced and formatted in Microsoft excel from this raw data.

Table 1 shows part of a sample *triangle* report. The report shows 1 million dollars spent *in* January 2010 *for* January 2010. *In* February 2010, $10,000 was retroactively spent *for* January 2010. *In* March, $5,000 was spent *for* January 2010. Thus, *through* March 2010, the total spent *for* January 2010 was $1,015,000, not $1,000,000. Similarly, through March, $1,650,000 was spent for February 2010. The name *triangle report,* comes from the intrinsic triangular formation.

	Jan 2010	Feb 2010	March 2010
January 2010	$1,000,000	$10,000	$5,000
February 2010		$1,500,000	$15,000
March 2010			$900,000

Table 1. Sample triangle report.

It is immediately apparent that the code producing the *proc means* summary for each of the 36 monthly payment files was similar, differing in the name of the file used for that particular month. Furthermore, obtaining the raw data for different contracts (call H# in the figures) and different review periods, requires adjusting three variables – *H, Y, M* – which stored the contract H#, the start year and start month respectively.

This is precisely a situation for which a CLIST is useful. *H, Y* and *M* can be obtained by user interaction at the terminal. The names of the 36 payment files can be code-generated using *H,Y* and *M*. The CLIST is also useful for checking whether each monthly file is available prior to processing the file.

BASIC ILLUSTRATIVE MODULES

Figure 2 illustrates a module communicating instructions to the user about what is needed to run the program. Figure 3 illustrates basic user input-output. Figure 4 illustrates basic arithmetic and string functions. Explanations of the code are contained in the figure captions. Some illustrations of how filenames depend on user-inputted values are given by the following: HKH.@BGD5050.PLNH1234.R082005,MONMEMD, HKH.@BGD5050.PLNH1234.R102005.MONMEMD, P#MMA.@BGD5050.PLNH1234.R012006.MONMEMD,HKH.@BGD5050.PLNH2345.R072004.MONMEM.

```
000100 SYSCALL INSTRUCTIONS
 ...
007700  INSTRUCTIONS: PROC 0
007800  K
007900  K
008000  WRITE HELLO USER: YOU MUST DO 4 THINGS PRIOR TO RUNNING PROGRAM
008100  WRITE -------- 1) YOU MUST INPUT AN H# - E.G. H1234
008200  WRITE -------- 2) YOU MUST INPUT A START YEAR - E.G. 2001
008300  WRITE -------- 3) YOU MUST INPUT A START MONTH - E.G. 1,3,11
008400  WRITE -------- 4) RUN 'DEARCHIVE' - ASCERTAIN FILE AVAILABILITY
008500  WRITE IF YOU DONT HAVE THIS INFO - COME BACK LATER
008600  WRITENR THANK YOU - KINDLY HIT ENTER TO CONTINUE
008700  READ &TEMP
008800  END
```

Figure 1. Sample code for a module with instruction communications to the user.

Figure 1 contains sample code illustrating a module communicating to the user. One *calls* the module in the main program using the *syscall* keyword. The code for the actual module is contained on lines 7700-8800. “K” clears the TSO screen. The *write* keyword is used to display instructions on the terminal. Lines 8100-8500 illustrate a bulleted, or menu, list of instructions. The *writenr* keyword on line 8600 is used when user input is expected which is stored in a dummy variable named *temp.* Line 7700 illustrates the module format: The module name *instructions,* followed by a colon, followed by the keyword *proc,* followed by an integer counting the number of parameters – in this case none – used by the module.

```
000300 SYSCALL USERINPUT
 ....
008900  /* ********************** USERINPUT ****************** */
009000  USERINPUT: PROC 0
009100  NGLOBAL H,Y,M
009200  K
009300  WRITENR ENTER THE H# IN FORM H1234 >>>>
009400  READ &H
009500  WRITENR ENTER THE YEAR IN THE FORM 2001 >>>>
009600  READ &Y
009700  WRITENR ENTER THE MONTH AS A NUMBER BETWEEN 1 AND 12 >>>>
009800  READ &M
009900  END
```

Figure 2. Sample code for a module requesting user input.

Figure 2 presents a module with sample code illustrating user-input. Line 300 illustrates calling the module in the main program. The actual module is found on lines 8900-9900. Line 8900 illustrates comments (/* */). CLIST is poor on *return values*; a workaround is using *global variables* declared on line 9100. The three user inputs are illustrated on lines 9300-9400, 9500-9600, and 9700-9800; the *writenr* keyword (lines 9300,9500,9700) is used to both write messages to the terminal and wait for user response. User response is stored in variables *H,Y,M* using the keyword *read.* Note: To properly store values, the ampersand (&) must precede the variable name similar to the *by-value vs. by reference* in several programming languages. The variables *H,Y,M* with the user-inputted information are now available throughout the program.

Figure 3 presents sample code that illustrates a *loop* (lines 600-800) calling a module (lines 11100 - 14000) illustrating basic CLIST arithmetic and string functions. The module takes one parameter (the counter from the loop) (line 11200) and "*returns*" one global variable, *FI* (line 11300), containing a filename. The filename depends on the loop counter and the initial year and date of the review period. The comments illustrate desired goals. If your start date is August 2005 and *I*=0 the *R* string should *R082005*; when *I*=1, the *R* string is *R092005*, when *I*=2, *R=R102005*; when *I*=6,*R=R022006*. The *if-then-else* statement, lines 12200-13100, using the *do-end* block structure, and the "+" *line continuation character* uses the *division* (/) and *remainder* (//) arithmetic operators (e.g. 25/12=2; 25//12=1). The setting of the *R* string value, lines 13200-13300, uses the *&Str* function to ensure that *variable values* (vs. actual *variable names*) are used. The actual mainframe filenames are set on lines 13400-13500 which illustrates the *period concatenation operator*. Vestiges of module testing are retained and illustrated as comments on lines 13800-13900.

SUBMISSION OF JCL and SAS CODE

We have already stated the key idea above: Write a basic JCL/SAS template and modify it for each payment month. The JCL/SAS code should produce as output the summary of the *proc means* procedure for that payment month with subtotals of payments for each allocated month. The CLIST accomplishes this task by communicating the values of certain JCL and SAS variables from CLIST variables.

Figured 4 and 5 show the basic template for submitting JCL with accompanying SAS within the CLIST. Both figures also illustrate communication between global CLIST variables and JCL and SAS variables.

The key idea in Figure 4 is contained in lines 93,94,202: Within the CLIST you edit a dummy file (line 93) by first clearing the file (line 94) and then writing your JCL and SAS code (see Figure 5). You then submit the newly written file (line 202), write, if desired, any messages to the user (line 203) and end the edit session without saving the newly written file (line 204). Lines 89,90 and 205 illustrate, as a condition for submitting the JCL, how to check file availability using the CLIST *sysdsn* function.

```
.000600 DO &I=0 TO 15
.000700  SYSCALL FILENAMECREATE &I
.000800 END
 ….
 011100  /* ****************** FILENAMECREATE *****************/
 011200  FILENAMECREATE: PROC 1 &J
 011300  NGLOBAL FI                  /* STORES FILENAME TO OPEN
 011400  /* *************** COMPUTE RELATIVE MONTH/YEAR  */
 011500  /*                                  */
 011600  /*        E.G. M=8,Y=2005   I=0===> R082005    */
 011700  /*                          I=4===> R122005    */
 011800  /*                          I=6===> R022006    */
 011900  /*                                  */
 012000  /* ****************************************  */
 012200  IF (&M+&J)//12 =0 THEN +
 012300    DO
 012400       SET &M1=12
 012500       SET &Y1=&Y +(&M+&J-1)/12
 012600    END
 012700  ELSE +
 012800    DO
 012900       SET &M1=(&M+&J)//12
 013000       SET &Y1= &Y+(&M+&J)/12
 013100    END
 013110  /* *************** COMPUTE R STRING ******************* */
 013120  /*                                  */
 013130  /*         E.G. R082005, R032006, R102005            */
 013140  /*                                  */
 013150  /* ************************************************** */
 013200  IF &M1 LT 10 THEN SET &R = R&STR(0)&STR(&M1)&STR(&Y1)
 013300  ELSE              SET &R = R&STR(&M1)&STR(&Y1)
 013310  /* *************** COMPUTE FILENAME ******************* */
 013320  /*                                  */
 013330  /*      2 FILENAMES DEPENDING ON BEFORE AFTER 2005      */
 013340  /*                                  */
 013350  /* ************************************************** */
 013400  SET &FI1 =HKH.@BGD5050.PLN&H..&R..MONMEMD
 013500  SET &FI2 =P#MMA.@BGD5050.PLN&H..&R..MONMEMD
 013600  IF Y1 GT 2005 THEN SET &FI = &FI2
 013700  ELSE               SET &FI = &FI1
 013800  /*  WRITENR &EVAL(&FI)>>>            USED DURING TESTING */
.013900  /*  READ &TEMP                     USED DURING TESTING */
.014000  END
```

Figure 3. Sample code illustrating loops, basic arithmetic and string functions.

Figure 5 complements and continues Figure 4. Figure 5 contains illustrative JCL (lines 95-114) and mainframe SAS code (lines 115-194). Only portions of the entire code are presented. Note the double numbering: the leftmost number is the *numbering within the CLIST* code while the rightmost number is the *JCL numbering* used in the submission to TSO. This illustrative code shows how the CLIST can take a basic JCL/SAS template and reuse it with minor modifications. Examples of the CLIST communicating with the JCL code are as follows: i) by changing the *IN1* filename (line 110); here *FI* was produced in the sample code of Figure 3; ii) by changing the values of the SAS variables *INPUTH, INPUTM* and *INPUTY* according to the user inputted information stored in the CLIST global variables, *H,M & Y* declared in the CLIST code of Figure 2 (line 120); by using the CLIST function, *sysdate*, to capture the system date and store it in a SAS variable, *temp1* (line 194).

CONCLUSION

We have reviewed CLISTS as a tool for user modularization, input-output, storage in global variables, manipulation of global variables with standard arithmetic and string functions, and editing and submission of JCL and SAS mainframe code. The tools presented in this paper, in conjunction with the user manuals cited in the references, should prove useful to programmers to immediately apply CLIST code to streamline tasks of intermediate capacity.

```
000089 IF &SYSDSN('&FI.') EQ OK THEN +
000090 DO
000093   EDIT  'H1C9.LIB.CNTL(DUMMY)' CNTL
000094   DELETE * 999999
……        /* Place here: JCL and MAINFRAME SAS programs (See Figure VI)
000202   SUBMIT
000203   WRITE YOU JUST DID MONTH &EVAL(&I+1) OF 36
000204   END NOSAVE
000205 END
```

Figure 4. A code shell for submission of JCL with SAS code (The code is continued in Figure 5).

```
000095 000010 //H1C9TRI JOB (FDAFDA8080,CO5),
000096 000020 //        H1C9TRI,
000097 000030 //        CLASS=B,
000098 000040 //        MSGCLASS=Q,
000099 000050 //        NOTIFY=H1C9
000108 000140 //S1 EXEC SAS9,WORK='20000,1000'
000109 000150 //*
000110 000160 //IN1 DD DSN='&FI.',DISP=SHR
000111 000170 //*
000112 000180 //OUT1 DD DSN=H1C9.@CLIST.P&I,DISP=OLD
000114 000200 //SYSIN DD *
000115 000210 DATA OUT1.A;
000116 000220 KEEP ...
000119 000233      PERIOD INPUTH INPUTM INPUTY;
000120 000236      INPUTH='&H'; INPUTM='&M'; INPUTY='&Y';
…
000194 000734 TEMP1='&SYSDATE';
…
```

Figure 5. Continuation of Figure 4. The figure illustrates submission of JCL and SAS code within a CLIST.

RECOMMENDED READING

IBM Reference manuals, http://ibmmainframes.com/manuals.php

IBM, z/OS V1R11.0 TSO/E CLISTS (Manual), Document Number, SA22-7781-05, http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/BOOKS/IKJ4B850/CCONTENTS

CONTACT INFORMATION

Russell Jay Hendel
7500 Security Boulevard
Baltimore, MD 21244
Phone: 410 786 0329
Russell.Hendel@cms.hhs.gov

Russell Jay Hendel
Dept. of Mathematics,
8000 York Road
Towson, MD 21252
Phone: 410 704 3091
RHendel@Towson.Edu, ProfessorHendel@GMail.Com