SAS® Code Generator based on Table-Driven Methodology in a Batch Environment

Arthur K. Yao
Bureau Of Labor Statistics, Washington DC

ABSTRACT

The main objective of the Table-driven System is to build a reliable batch application that is easy to develop and maintain. Most batch programs are treated as a black box with long lines of repetitive code that is very difficult to maintain. The Table-driven system will allow subject matter specialist to easily change system variables and values without requiring code changes. The type of batch applications, that the Table-driven System is best suited, are applications that have simple repetitive logic with different input parameters.

The Table-driven System consists of three main components: a conceptual table (Edit Table), a corresponding macro code to support the conceptual table (Edit Macro), and the Code Generator Macro to piece it all together in a SAS® Data Step.

The Edit Table consists of Business Rules, and each Edit Table requires a corresponding Edit Macro to translate the business rules into SAS code. The Code Generator Macro uses both the Edit Table and the Edit Macro as input to produce a SAS Data Step with the appropriate SAS code to perform the edits. Given the present architecture, it is possible to maintain the Edit Table at a conceptual level versus code level that meets our objective.

BACKGROUND

The Consumer Expenditure Survey program provides a continuous and comprehensive flow of data on the buying habits of American consumers for use in a wide variety of economic research and in support of periodic revisions of the Consumer Price Index. The Consumer Expenditure Systems Branch (CE) supports and develops computer systems for the Consumer Expenditure Survey program. The majority of applications are coded using SAS in a batch environment. These SAS applications deal with many types of imputation and allocation of consumer expenditure data. It also requires extensive number crunching capabilities on a large set of data with minimal user intervention. The business rules range from simple range checks to complex imputation and allocation based on a variety of Statistical methodologies. The majority of application systems are developed and maintained on the Mainframe (IBM® MVS Environment) with an eye towards the downsized environment (e.g., UNIX and Windows environment).

The front end of the Table-driven system (Edit Tables) is developed on the Microsoft® Windows environment using Microsoft EXCEL. This decision was influenced by the familiarity of using EXCEL software by our internal users who are responsible for maintaining the Edit Tables. Our internal users are Economist and Statisticians, while our external users are private and public institutions, and the general public.

WHAT IS AN EDIT TABLE?

An Edit Table is a conceptual table that contains the business rules to edit the data. It is the source of information where SAS code is to be generated within a data step. Each row of the table represents a block or section of SAS code; while each column of the table represents user specified token. The token serves two purposes: 1) To dictate which tokens to use to generate SAS code. 2) To be substituted as part of a SAS code.

TYPE	NAME	COND	VALU	FMTN
1	20	200	20	8
1	X	0<X<1		BUSPT
2	Y Z	Y+Z >1		BUSPT
3	A	B >0 & A >B	B	

Fig. 1

The following Edit Table in Fig. 1 shows an example of three types of imputation to perform requiring five columns. Only the first two rows are non-data rows. The first row indicates the header of each column while the second row indicates the maximum length allowed for the particular column. The maximum length is required when the Edit Table transforms into a SAS dataset where the first rows represent the SAS variable name, and the second row represents the length of each SAS variable. The columns of the Edit Table are user defined. The first Column TYPE represents the different types of imputation being performed. The second Column NAME represents the variable name(s) to be processed. The third column COND represents the condition criterion for imputation. The fourth column VALU represents the value to be assigned. Lastly, the fifth column FMTN represents the format name used to impute the variable. Please note that these five columns are all arbitrarily chosen by the programmer for their individual Table-driven programs. The three data rows of the Edit Table above can be translated into three general business rules:

For all <TYPE> = 1:

If <COND> then assign <NAME> based on the <FMTN> format.

For all <TYPE> = 2:

If (<NAME[1]> > 0 & <NAME[2]> > 0 & <COND>) then assign both <NAME[1]> and <NAME[2]> based on the <FMTN> format.

For all <TYPE> = 3:

If (<COND>) then assign <NAME> = <VALU>.

The physical attribute of the Edit Table in our current implementation encompasses three different types of file formats during its life time as shown below in Fig. 2.

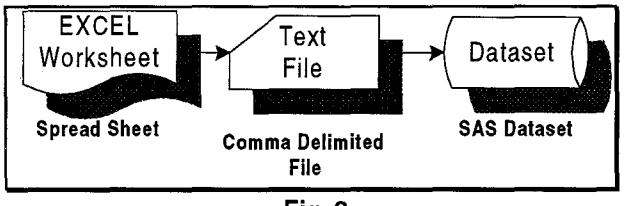

Fig. 2

The transformation from an EXCEL Worksheet into a SAS dataset requires three steps in our current implementation. The first step of the transformation process is to take the EXCEL Worksheet as shown in Fig. 1, and save as a comma delimited text file (*.CSV format) as shown in Fig. 3.

```
TYPE,NAME,COND,VALU,FMTN
1,20,200,20,8
1,X,0<X<1,,BUSPT
2,Y Z,Y+Z>1,,BUSPT
3,A,B>0 & A>B,B,
```

Fig. 3

The second step of the process is to upload the ASCII text file onto the mainframe (IBM MVS) environment. The third and last step of the transformation process is to convert the comma delimited text file into a SAS dataset via SAS code as shown in Fig. 4 using the PROC PRINT procedure in SAS.

EDIT TABLE DATASET (EDITTABL)

OBS	TYPE	NAME	COND	VALU	FMTN
1	1	X	0<X<1		BUSPT
2	2	Y Z	Y+Z>1		BUSPT
3	3	A	B>0 & A>B	B	

Fig. 4

The converted SAS dataset named EDITTABL is the final state of the Edit Table before being processed in conjunction with the Edit Macro.

WHAT IS AN EDIT MACRO?

An Edit Macro implements the Business rules in the Edit Table. The Edit Macro is a SAS Macro code that is tailored to a given Edit Table. Given the header of the Edit Table, the Edit Macro must have the same parameters defined as the header. In our given example of the Edit Table (See Fig. 1), the parameter of the Edit macro will be: TYPE, NAME, COND, VALU, FMTN. The Edit Macro manipulates a given Edit Table using SAS Macro code to generate SAS data step code. For each row of the Edit Table, the entire Edit Macro is executed in the SAS Macro facility. The following is an example of the Edit Macro named %EDITMAC in Fig. 5 for the corresponding Edit Table in our example.

```
%MACRO EDITMAC (TYPE =, NAME =, COND =, VALU =,
                FMTN =);
 /* DECLARE LOCAL MACRO VARIABLES FOR TYPE = 2 */
 %LOCAL TEMPVAR1 TEMPVAR2;

 %IF (&TYPE = 1) %THEN
  %DO; /* TYPE ONE */
   IF (&COND) THEN
    DO; /* TYPE ONE IMPUTATION */
     &NAME = INPUT(PUT("&NAME", $&FMTN..),4.);
    END; /* TYPE ONE IMPUTATION */
  %END; /* TYPE ONE */
 %ELSE %IF (&TYPE = 2) %THEN
  %DO; /* TYPE TWO */
   %LET TEMPVAR1 = %SCAN(&NAME,1,%STR( ));
   %LET TEMPVAR2 = %SCAN(&NAME,2,%STR( ));
   IF (&TEMPVAR1 > 0 & &TEMPVAR2 > 0 & &COND) THEN
    DO; /* TYPE TWO IMPUTATION */
     &TEMPVAR1 =INPUT(PUT("&TEMPVAR1",$&FMTN..),4.);
     &TEMPVAR2 =INPUT(PUT("&TEMPVAR2",$&FMTN..),4.);
    END; /* TYPE TWO IMPUTATION */
  %END; /* TYPE TWO */
 %ELSE %IF (&TYPE = 3) %THEN
  %DO; /* TYPE THREE */
   IF (&COND) THEN
    DO; /* TYPE THREE IMPUTATION */
     &NAME = &VALU;
    END; /* TYPE THREE IMPUTATION */
  %END; /* TYPE THREE */
%MEND EDITMAC;
```

Fig. 5

The main body of the Edit Macro %EDITMAC in Fig. 5 would be similar to the three general business rules stated above. The only information that has not yet been furnished by the Edit Macro and the Edit Table is the format referenced for TYPE = 1 & 2. The pre-defined SAS character format: BUSPT would be declared prior to the actual execution of the Code Generator Macro %GENCODE. An example of the BUSPT format defined is illustrated in Fig. 6 where each variable X, Y, and Z would have a corresponding pre-designated value to assign to each variable.

```
PROC FORMAT;
VALUE $BUSPT
 'X' = '0.70'
 'Y' = '0.80'
 'Z' = '0.90';
RUN;
```

Fig. 6

Type one and three are straight translation from the business rules into SAS Macro code while type two

requires extra SAS Macro code to separate the two variable lists in the NAME column. The two variable names are separated and stored in temporary local macro variables for processing. This is a straight forward example of a relatively simple Edit Macro code that performs three basic types of imputation on the in-coming data.

WHAT IS A CODE GENERATOR MACRO?

A Code Generator Macro is the Engine behind the entire Table-driven system. The Code Generator Macro is designed to generate a data step and the SAS code within a data step for processing. It takes as it's input the Edit Table dataset name, the Edit Macro code name, the input and output dataset name for the data step, and the list of the columns of the Edit table for processing as shown in Fig. 7 below:

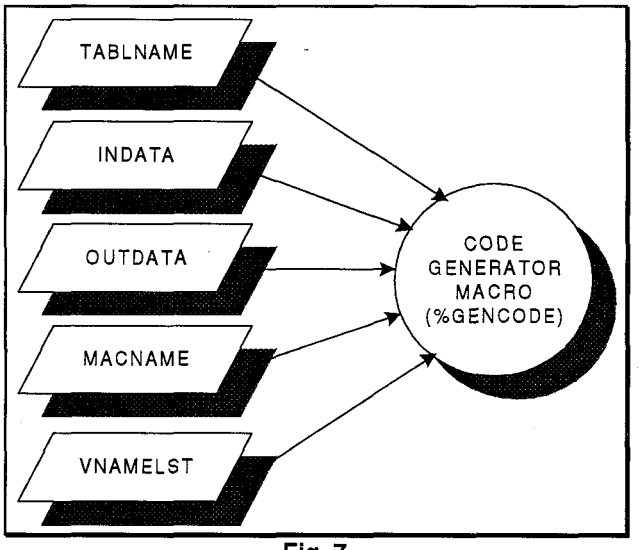

Fig. 7

The TABLNAME parameter is the SAS dataset name of the Edit Table. The INDATA parameter is the input dataset name of actual Consumer Expenditure (CE) data to be processed. The OUTDATA parameter is the output dataset name of the processed data. The MACNAME parameter is the macro name of the Edit Macro. Finally, the VNAMELST parameter is the list of column names of the Edit table used to generate local macro variables. The name of the Code Generator Macro is called %GENCODE. Given the input CE data dataset is named CEINDATA, and the processed output CE data dataset is named CEOUTDAT, the following is a typical call of the macro %GENCODE given our example.

```
%GENCODE (TABLNAME = EDITTABL,
          INDATA  = CEINDATA,
          OUTDATA = CEOUTDAT,
          MACNAME = EDITMAC,
          VNAMELST = TYPE NAME COND VALU FMTN)
```

Fig. 8

The content of the Code Generator Macro %GENCODE is illustrated in Fig. 9. The %GENCODE macro can be dissected into two parts for ease of understanding. The first part is to generate a temporary local macro variable

```
%MACRO GENCODE (TABLNAME =, INDATA =, OUTDATA =,
                MACNAME =, VNAMELST = );
 /* DECLARE LOCAL MACRO VARIABLES */
 %LOCAL GENCODE1 GENCODE2 GENCODE3
        VNAME VNAMENUM VARNUMAL;
 /*VARNUMAL CONTAINS LIST OF ALL TYPE NUM VARS */
 %LET VARNUMAL = ; /* INITIALIZE LIST TO NULL */
 %IF (&INDATA ^= _NULL_) %THEN %DO;
  /* OBTAIN VARIABLES AND THEIR ATTRIBUTE TYPES */
  PROC CONTENTS DATA = &INDATA
       OUT = CONTENT (KEEP = NAME TYPE)
       NOPRINT;
  RUN;
  /* GENERATE LOCAL MACRO VARS FOR NUM ONLY. */
  DATA _NULL_;
   SET CONTENT END = LASTREC;
   LENGTH COUNTNUM 4 VARNAME $8; /* SIZE OF VARS */
   RETAIN COUNTNUM 1; /* INIT. COUNTER TO ONE */
   IF TYPE = 1 THEN DO; /* VARIABLE IS NUMERIC */
    VARNAME = "VNUM" || TRIM(LEFT(PUT(COUNTNUM,4.)));
    CALL SYMPUT(VARNAME,NAME); /* CREATE MAC VAR */
    COUNTNUM + 1; /* INCREMENT THE COUNTER */
   END;
   IF LASTREC THEN DO;
   /* STORE TOTAL NUMBER OF VARS INTO A MAC VAR */
    COUNTNUM = COUNTNUM - 1;
    CALL SYMPUT("GENCODE1",PUT(COUNTNUM,4.));
   END;
  RUN;
  /* CONSTRUCT ONE LIST WITH NAME OF ALL NUMERIC */
  /* VARIABLES. USING SPACE AS A DELIMITER.    */
  %DO I = 1 %TO &GENCODE1;
   %LET VARNUMAL = &VARNUMAL &&VNUM&I;
  %END;
  /* SET FINAL LIST TO UPPERCASE AND ADD BLANKS */
  /* TO BOTH END OF THIS LIST. THIS WILL PREVENT */
  /* INDEXING OF A SUBSTRING.                   */
  %LET VARNUMAL = %UPCASE( &VARNUMAL );
 %END; /* IF INDATA IS NOT NULL */

 %NOBS (DATA = &TABLNAME)
 %LET VNAMENUM = %WORDCNT(LIST = &VNAMELST);

 /* GENERATE LOCAL MAC VARS BASED ON INPUT TABLE */
 DATA _NULL_;
  SET &TABLNAME;
  OBSNUM = TRIM(LEFT(PUT(_N_,32.)));
  %DO GENCODE2 = 1 %TO &VNAMENUM;
   %LET VNAME=%SCAN(&VNAMELST,&GENCODE2,%STR( ));
   CALL SYMPUT("&VNAME"||OBSNUM,&VNAME);
  %END; /* %DO LOOP */
 RUN;

 /* CALL THE TEST MACRO WITHIN A DATA STEP */
 DATA &OUTDATA;
  SET &INDATA;
  %DO GENCODE2 = 1 %TO &NOBS;
   %&MACNAME (
   %DO GENCODE3 = 1 %TO &VNAMENUM;
    /* GENERATE PARAMETERS OF THE MACRO CALL */
    %LET VNAME=%SCAN(&VNAMELST,&GENCODE3,%STR( ));
   &VNAME = &&&VNAME&GENCODE2
    %IF &GENCODE3 ^= &VNAMENUM %THEN %DO;
    /* IF NOT LAST PARA THEN PUT COMMA SEPARATOR */
                ,
    %END;
   %END; /* %DO LOOP */
         )
  %END; /* %DO LOOP */
 RUN;
%MEND GENCODE;
```

Fig. 9

that contains the list of all the numeric variables within the input CE data (CEINDATA) dataset. The numeric variable list is used as a look-up table to differentiate between a character variable and a numeric variable when generating code for the respective data types. It is only used by the Edit Macro when a comparison of variable to a constant is needed where the attribute of the variable is unknown until run time. Under normal circumstances, SAS will automatically convert the attribute type for comparison and leave a note on the SAS Log. The note on the SAS Log would state that a conversion of type numeric or character has been done at a specific location of the code. The SAS Note statement for the automatic conversion is contrary to the CE office coding standards document which states in short that an automatic attribute type conversion by SAS is not allowed. All attribute type conversion should be explicitly stated in code by the programmer. In our example, we do not need to use the numeric variable list; therefore there is no reference to the local macro variable name VARNUMVAL in our Edit Macro %EDITMAC.

The second part of the %GENCODE macro starts with the call to the macro %NOBS. The second part is the main section of the code generator. The two SAS macros: %NOBS and %WORDCNT referenced in %GENCODE are general system utility macros coded by our own CE programmers. These two utility macros are fairly simple and short macros that perform a specific function. %NOBS counts the number of observations of a SAS dataset and stores the return value into a GLOBAL macro variable called NOBS. While %WORDCNT is a function that counts the number of variables in a given macro variable list delimited by spaces, and simply return the value back to the SAS program. Due to the simplicity of these two macros, the source codes for the two macros are not provided.

After %GENCODE determines how many observations, the column's name and the number of columns in the Edit table, the generation of local macro variables can than commence within the Data Null step. For each cell (row, column) of the Edit table, a local macro variable is generated to store the Edit table information. The naming convention of these local macro variables is done using the column header concatenated with a counter as its subscript. Notice that each header of the Edit table only has a maximum length of 4 characters long. This limitation was put on the Edit table header naming convention so that the subscript counter has a range from "1" to "9999" respectively given eight characters is the upper limit for a SAS macro variable.

In our example, the Data Null step will automatically generate 15 (5 X 3) local macro variables and store the content of each cell in its respective macro variable. Fig. 10 represents the conceptual view of this Data Null process given our Edit table. The SAS local macro variable naming convention would be:

<Column Header><n (Where n = 1-9999)>

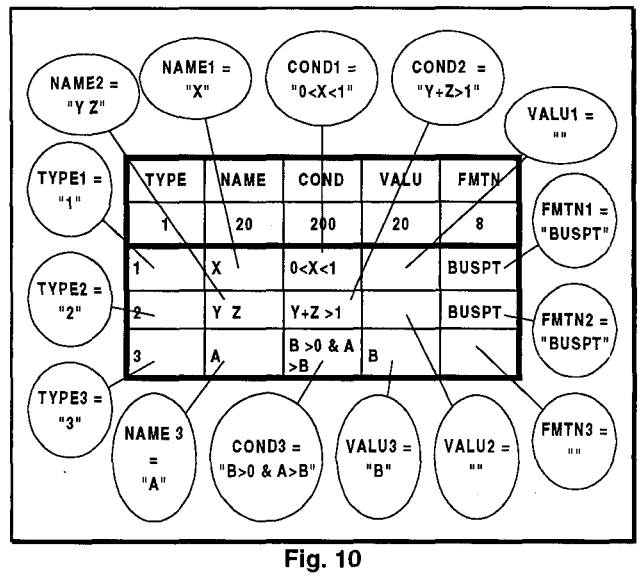

Fig. 10

In Fig. 11 below illustrate the SAS Log generated after the execution of %GENCODE for the Data Null step.

```
.
.
MPRINT(GENCODE): DATA _NULL_;
MPRINT(GENCODE): SET EDITTABL;
MPRINT(GENCODE): OBSNUM = TRIM(LEFT(PUT(_N_,32.)));
MPRINT(GENCODE): CALL SYMPUT("TYPE"||OBSNUM,TYPE);
MPRINT(GENCODE): CALL SYMPUT("NAME"||OBSNUM,NAME);
MPRINT(GENCODE): CALL SYMPUT("COND"||OBSNUM,COND);
MPRINT(GENCODE): CALL SYMPUT("VALU"||OBSNUM,VALU);
MPRINT(GENCODE): CALL SYMPUT("FMTN"||OBSNUM,FMTN);
MPRINT(GENCODE): RUN;
.
.
```

Fig. 11

```
.
.
/* CALL THE TEST MACRO WITHIN A DATA STEP */
 DATA CEOUTDAT;
  SET CEINDATA;
  %DO n = 1 TO 3;
   %EDITMAC (TYPE = &&TYPE&n,
             NAME = &&NAME&n,
             COND = &&COND&n,
             VALU = &&VALU&n,
             FMTN = &&FMTN&n)
  %END; /* DO LOOP */
 RUN;
```

Fig. 12

The last Data step in %GENCODE is the actual Data step where the calculation of CE data is performed. To explain the last Data step more clearly, we partially resolved some local macro variables and renamed the counter macro variable in Fig. 12 based on our example. It is then clear to see that the counter variable "n" is the row index of the Edit table, and the Edit macro %EDITMAC is executed for each row of the Edit table. The parameters being passed into the Edit macro

%EDITMAC would resolve into the content of the 15 local macro variable as shown in Fig. 10 respectively.

```
.
.
MPRINT(GENCODE): DATA CEOUTDAT;
MPRINT(GENCODE): SET CEINDATA;
MPRINT(EDITMAC): IF (0<X<1) THEN DO;
MPRINT(EDITMAC): X = INPUT(PUT("X", $BUSPT.),4.);
MPRINT(EDITMAC): END;
MPRINT(EDITMAC): IF (Y>0 & Z>0 & Y+Z>1) THEN DO;
MPRINT(EDITMAC): Y = INPUT(PUT("Y", $BUSPT.),4.);
MPRINT(EDITMAC): Z = INPUT(PUT("Z", $BUSPT.),4.);
MPRINT(EDITMAC): END;
MPRINT(EDITMAC): IF (B>0 & A>B) THEN DO;
MPRINT(EDITMAC): A = B;
MPRINT(EDITMAC): END;
MPRINT(GENCODE): RUN;
.
.
```

Fig. 13

The SAS Log in Fig. 13 is generated after the execution of %GENCODE for the Data step in Fig. 12. The code generated has three general sections of code that maps directly to the three rows in the Edit table. It is strongly recommended that the programmer should know the kind of code to generate before composing their Edit Macro code. It is critical that the design of the Edit table is easy to maintain and understand by both the immediate user and the programmer. Therefor the design of the Edit Table should be approved by the user and the programmer before actual implementation of the particular Table-driven edit.

TEST RESULTS

To verify the logic of our example, we will use the following test data for our acceptance testing. Fig. 14 is the printout of the input SAS dataset (CEINDATA) before the edit is performed.

TEST INPUT DATASET (CEINDATA)

OBS	X	Y	Z	A	B
1	0.5	0.0	0.0	0.0	0.0
2	0.0	0.6	0.7	0.0	0.0
3	0.0	0.0	0.0	0.4	0.2

Fig. 14

The test data is designed such that the first observation is to test TYPE = 1 case, while the second and third observations are test cases for TYPE = 2 and 3 respectively. Fig. 15 shows the results after the input data have been edited via Table-driven system.

According to our logic for TYPE = 1 case: If the variable X is between zero and one exclusively, then reassign the value of X according to the BUSPT format which in this case is 0.70 (X = 0.7). For TYPE = 2 case: If both variables Y and Z are between zero and one exclusively and the sum of Y and Z is greater than one, then reassigns the value of X according to the BUSPT format. In this instance, Y is 0.80 (Y = 0.7) and Z is 0.90 (Z = 0.9). For TYPE = 3 case: If the variable B is greater than zero and the variable A is greater than B, then reassigns the value of A to the value of B (A = B). The resulting data in this instance verifies the Edit Table and the Edit Macro did perform the three types of imputation correctly.

TEST OUTPUT DATASET (CEOUTDAT)

OBS	X	Y	Z	A	B
1	0.7	0.0	0.0	0.0	0.0
2	0.0	0.8	0.9	0.0	0.0
3	0.0	0.0	0.0	0.2	0.2

Fig. 15

TABLE DESIGN AND MAINTENANCE

One example of table design for better maintenance is to allows the flexibility for business rules to change over time without re-creating another Edit table for the same edit. Four extra columns is added at the end of each Edit Table as a general standard. These last four rows contain the start and end date for when the row of data is active. The start date (Year and Month) and an end date are kept to reflect the validity of the business rule for the given period of time as illustrated in Fig. 16.

...	STAMO	STAYR	ENDMO	ENDYR
...	2	4	2	4
...	01	1994	03	1995
...	04	1995	-	-
...	01	1994	-	-

Fig. 16

Both the old and new business rules can coexist in the same table without ambiguity or confusion. It acts as a versioning control device to allow Re-Running of a Table driven Edit without manual intervention. Each row of the Edit Table will be used only when the period of processing is in between the start and end date exclusively given that the CE processing period is done by monthly. A dash "-" in the ENDMO and ENDYR column represents the most current row of data to be used for the Table-driven system. The SAS code to perform the filtering out process is done during the conversion of the comma delimited text file into a SAS dataset. Note that these four columns do not have to be four characters long like the rest of the columns in the Edit Table. Once the filtering process is performed on the Edit Table, the last four SAS dataset variables: STAMO, STAYR, ENDMO, and ENDYR are dropped from the converted SAS dataset; therefore the code generator

macro %GENCODE will never see the four house-keeping variables.

CONCLUSION

Given the illustration and description of all the bare essential tools and utilities one will need, one can construct a reliable Table-driven application that is easy to develop and maintain. There is still much room left for enhancements and improvements to the current system based on many factors: the nature of the business rules, the type of applications needed, and many design implications. The CE Table-driven system presented here is approximately one-third of the entire CE Table-driven system that encompasses data extraction and format or look-up table designs. There could be many more possibilities for the use of this Table-driven methodology. We hope future programmers and analysts would find more innovative way to utilize this Table driven system.

This system is aimed toward legacy batch applications with simple well-defined business rules where maintenance is at a premium. We see this Table-driven system will bring the programmer and the user more closely together in developing and maintaining future batch applications.

ACKNOWLEDGMENTS

The author thanks all those involved in consulting and building this Table-driven system. Special thanks to Richard Schroeder, Branch Chief for Consumer Expenditure Survey Systems Branch, for his continuing support and commitment to the Table-driven concept to build one of our major systems within the CE Branch, and without whom this Table-driven system and paper would not be possible. The author may be contacted at:

Arthur K. Yao
Bureau Of Labor Statistics
2nd Massachusetts Avenue NE,
PSB Bldg, Rm 5935
Washington DC 20212-0001

PH: (202) 606-6729
FAX: (202) 606-6772
Email: yao_a@cpi.po1.bls.gov

TRADEMARKS