

# Cramped for Drive Space? Save Space with the Auto-Compress Macro

Adam Bemis, Household International, Beaverton, OR  
Scott Hanson, Household International, Beaverton, OR

## ABSTRACT

Although storage space is relatively cheap these days, it may still be a scarce commodity. To help remedy this problem, SAS offers three options for compressing SAS datasets: No compression, character compression, or binary compression. However, choosing and sticking with one option will never be optimal for all datasets, because optimality depends upon the number of numeric and character variables as well as the number of observations. Unfortunately, SASv8 does not have a system option that automatically chooses the optimal compression technique; instead the user must figure this out. It is a simple but tedious task; it requires testing each compression option on a dataset and reading the resulting file sizes, then selecting the smallest file. Fortunately, the Auto-Compress Macro will do this for you; it will optimally compress a dataset, a library, or an entire directory (with a Visual Basic for Application (VBA) interface) and provide statistics of the datasets it attempted to compress.

## INTRODUCTION

Remember Y2K? Drive space in the 1960s was extremely expensive; therefore, in order to save space, programmers coded years in a two-digit field. Today, drive space is much cheaper and saving disk space may not be on the forefront of a programmers mind, but the reality is that an institution will probably not be willing to provide an unlimited amount of disk drive resources. Such is the case at Household International; despite a nearly full server, acquiring more drive space is almost impossible. Processes would crash because the server was at capacity and employees would be asked to clean up their domain to free up drive space only to have the server filled to capacity again. This vicious cycle repeated itself several times.

Many times over, frustrated employees tried to resolve this problem, but these attempts were thwarted by a lack of cooperation. As Household International is a financial institution, most of the server was filled with large SAS datasets, comprising millions of observations and hundreds of variables. SAS dataset compression was one of the suggested remedies to the problem, but few employees were consistently using it, especially to go back and compress older files. As a result, the Auto-Compress Macro was invented to automatically compress SAS datasets to their optimal compression.

## Foreword on SAS Compression

The purpose of this article is to discuss the Auto-Compress Macro, however, before using the macro users should be aware of the pros and cons of SAS compression. SAS has three options for compressing SAS datasets: No compression (SAS default), character compression or binary compression. The primary reason for using SAS compression is to reduce the required storage space of a SAS dataset. This reduction in space also decreases I/O, which may reduce overall processing time to finish a job, because the files that are read in and written out are smaller. The primary drawback to compression is an increase in CPU processing time because SAS uncompresses the dataset to process it, then recompresses it. SAS compression may also result in a file larger than an uncompressed file if the SAS

dataset attributes are not appropriate for the compression algorithm. For more details on SAS compression, the authors suggest additional reading on the topic.

## MACRO DESIGN

The Auto-Compress Macro is designed to iterate through multiple libraries and evaluate each SAS dataset within the library. If the macro determines that a compression algorithm would result in a smaller file size the macro compresses the file. Although the macro can be run directly from SAS, it was designed to be run using a VBA based GUI from Excel. The GUI interface requires little SAS knowledge; the user simply points to a directory and submits the job.

The macro is broken down into three components or individual macros: a macro to compress datasets, a macro to read all the datasets in a library, and a macro to assign libraries to all the directories in a root path. The components of each macro will be discussed in detail.

## Compress Datasets Macro

The compress datasets macro is the core of the Auto-Compress macro: it evaluates each dataset and determines key information about the dataset, including its current compression. The macro then evaluates the dataset to see if it can be compressed, and if it can, the macro performs the data compression.

There are four inputs to the compress datasets macro: LIBRARY, DSN, DTDNCBFR, and DTDNCAFTR. Both LIBRARY and DSN are required and refer to a valid library reference and dataset name, respectively. The other inputs are optional and may be omitted. They may be useful when compressing libraries or entire directories. DTDNCBFR stands for "date do not compress before" and DTDNCAFTR stands for "date do not compress after." Both the dates refer to file modification dates; this can be useful if the user does not want to attempt to compress files that have already been compressed by the macro, or to compress SAS datasets that are currently being used. If left blank, the values are set to defaults allowing the macro to attempt to compress all files. The %GLOBAL statement enables the macro to pass macro variable values to the Compress Library Macro.

```
%MACRO COMPRESS_DATASET (LIBRARY , DSN , DTDNCBFR ,  
    DTDNCAFTR ) ;  
%GLOBAL CURRCOMP COMPRESSION NUMOBS NUMVAR  
    NUMRCVAR ;  
%IF &DTDNCBFR EQ %THEN %LET DTDNCBFR = 0 ;  
%ELSE %LET DTDNCBFR =  
    %SYSFUNC ( INPUTN ( &DTDNCBFR , MMDDYY10 . ) ) ;  
%IF &DTDNCAFTR EQ %THEN %LET DTDNCAFTR = 999999 ;  
%ELSE %LET DTDNCAFTR =  
    %SYSFUNC ( INPUTN ( &DTDNCAFTR , MMDDYY10 . ) ) ;
```

The next piece of the macro gathers pertinent information about the dataset by creating an output dataset from a PROC CONTENTS. The gathered data is passed to macro language via a DATA \_NULL\_ and CALL SYMPUT statement for later reference. There are also some error checking features that ensure that the proc contents is successful and that the output

CNTNTS dataset is not empty. Upon failure, the macro jumps to the end of the macro and the error is noted in the log.

```

PROC CONTENTS DATA=&LIBRARY..&DSN NOPRINT
  OUT=CNNTNTS; RUN;
%IF &SYSERR NE 0 %THEN %DO;
  %PUT FAILED TO CREATE CONTENTS FOR
    &LIBRARY..&DSN;
  %LET CURRCOMP = FCNTNT;
  %LET COMPRESSION = FCNTNT;
  %LET NUMOBS = .;
  %LET NUMVAR = .;
  %LET NUMRCVAR = .;
  %GOTO STOPLOOPING;
%END;
DATA _NULL_;
  IF 0 THEN SET CNTNTS NOBS=NOBS1;
  CALL SYMPUT("NOBS",COMPRESS(PUT(NOBS1,12.)));
  STOP;
RUN;
%IF &NOBS EQ 0 %THEN %DO;
  %PUT FAILED: &LIBRARY..&DSN CONTAINS NO
    VARIABLES;
  %LET CURRCOMP =FNOVAR;
  %LET COMPRESSION =FNOVAR;
  %LET NUMOBS = 0;
  %LET NUMVAR = 0;
  %LET NUMRCVAR = 0;
  %GOTO STOPLOOPING;
%END;
DATA _NULL_ ;
  SET CNTNTS END=LAST;
  RETAIN NUMRCVAR;
  IF TYPE EQ 1 THEN NUMRCVAR + 1;
  IF LAST THEN DO;
    IF COMPRESS EQ "YES" AND ENGINE NE "V6"
      THEN COMPRESS = 'CHAR';
    CALL SYMPUT("CURRCOMP",TRIM(COMPRESS));
    CALL SYMPUT("DTMOD",
      COMPRESS(PUT(DATEPART(MODATE),12.)));
    CALL SYMPUT("ENGINE",TRIM(ENGINE));
    CALL SYMPUT("NUMOBS",
      COMPRESS(PUT(NOBS,12.)));
    CALL SYMPUT("NUMRCVAR",
      COMPRESS(PUT(NUMRCVAR,12.)));
    CALL SYMPUT("NUMVAR",
      COMPRESS(PUT(_N_,12.)));
  END;
RUN;

```

The next section runs if the file modification date falls within the specified range. This section estimates optimal compression by creating a temporary dataset of the original dataset using each compression algorithm. This temporary dataset is limited by a specific number of observations; the purpose is to create a file with a size of at least 1 MB. Error checking is also performed to ensure completion of each data step. Using the output delivery system (ODS), the dataset FSIZE is created from a PROC DATASETS. This step provides a field, FILE\_SIZE, which is used to determine what compression algorithm results in the smallest file. The result is then passed to macro language.

```

%IF &DTDNCFR LE &DTMOD AND &DTMOD LE &DTDNCFTR
  %THEN %DO;
%IF &&NUMVAR LE 5 %THEN %LET OBS=50000;
%ELSE %IF &&NUMVAR LE 10
  %THEN %LET OBS=35000;
%ELSE %LET OBS=10000;
DATA C_NO(COMPRESS=NO);
  SET &LIBRARY..&DSN(OBS=&OBS);
RUN;
%IF &SYSERR NE 0 %THEN %DO;
  %PUT FAILED TO CREATE C_NO FOR

```

```

  &LIBRARY..&DSN;
  %LET COMPRESSION = FAILNO;
  %GOTO STOPLOOPING;
%END;
DATA C_CHAR(COMPRESS=CHAR);
  SET &LIBRARY..&DSN(OBS=&OBS);
RUN;
%IF &SYSERR NE 0 %THEN %DO;
  %PUT FAILED TO CREATE C_CHAR FOR
    &LIBRARY..&DSN;
  %LET COMPRESSION = FAILCH;
  %GOTO STOPLOOPING;
%END;
%IF &ENGINE NE V6 %THEN %DO;
  DATA C_BINARY(COMPRESS=BINARY);
    SET &LIBRARY..&DSN(OBS=&OBS);
  RUN;
  %IF &SYSERR NE 0 %THEN %DO;
    %PUT FAILED TO CREATE C_BINARY FOR
      &LIBRARY..&DSN;
    %LET COMPRESSION = FAILBI;
    %GOTO STOPLOOPING;
  %END;
%END;
ODS OUTPUT DATASETS.MEMBERS=WORK.FSIZE;
  PROC DATASETS LIB=WORK MEMTYPE=DATA; QUIT;
ODS OUTPUT CLOSE;
DATA _NULL_ ;
  SET WORK.FSIZE(WHERE=(MEMNAME IN ("C_NO",
    "C_CHAR","C_BINARY"))) END=LAST;
  LENGTH COMPTYPE $6;
  RETAIN SMALLEST COMPTYPE;
  IF _N_ EQ 1 THEN DO;
    SMALLEST = FILE_SIZE;
    COMPTYPE = SUBSTR(MEMNAME,3,
      LENGTH(MEMNAME)-2);
  END;
  ELSE IF FILE_SIZE LT SMALLEST THEN DO;
    SMALLEST = FILE_SIZE;
    COMPTYPE = SUBSTR(MEMNAME,3,
      LENGTH(MEMNAME)-2);
  END;
  IF LAST THEN CALL SYMPUT("COMPRESSION",
    TRIM(COMPTYPE));
RUN;

```

The last section of the macro compares the dataset's current compression to the optimal compression determined by the macro; if the dataset's compression is not optimal, it is compressed. Again, a step exists to determine if compression is successful or not. The last part of the macro simply cleans up the temporary work files the macro created.

```

%IF &COMPRESSION EQ CHAR AND &ENGINE EQ V6
  %THEN %LET COMPRESSION = YES;
%IF &CURRCOMP NE &COMPRESSION %THEN %DO;
  %PUT #####;
  %PUT ## COMPRESSING DATASET &LIBRARY..&DSN
    USING &COMPRESSION COMPRESSION.;
  %PUT #####;
  DATA &LIBRARY..&DSN
    (COMPRESS=&COMPRESSION);
  SET &LIBRARY..&DSN;
RUN;
%IF &SYSERR NE 0 %THEN %DO;
  %PUT FAILED TO COMPRESS FILE
    &LIBRARY..&DSN;
  %LET COMPRESSION = FCOMP;
  %GOTO STOPLOOPING;
%END;
%END;
%ELSE %LET COMPRESSION = DTXCLD; *DATE EXCLUDED;
%STOPLOOPING;

```

```
PROC DATASETS LIB=WORK NOLIST;
  DELETE CNTNTS C_BINARY C_CHAR C_NO FSIZE;
QUIT;
%MEND COMPRESS_DATASET;
```

### Compress Library Macro

The Compress Library Macro creates a SAS datasets that lists each dataset in the library; then it calls the Compress Datasets Macro for each dataset. It has three inputs: LIBRARY, DTDNCBFR, and DTDNCAFTR. Library represents a library reference that will be compressed by the macro. The two dates are optional and are the same as previously described.

The list file, FILELISTER, is created by using the ODS and PROC DATASETS to create a dataset. Each dataset name is then passed to macro processing via the CALL SYMPUT statement in a DATA \_NULL\_. If the PROC DATASETS fails or no SAS datasets exist in the library then the macro skips to the end of the macro.

```
%MACRO COMPRESS_LIBRARY(LIBRARY,DTDNCBFR,
  DTDNCAFTR);
%LET NODATASETS = 0;
%LOCAL I;
ODS OUTPUT DATASETS.MEMBERS=WORK.FILELISTER;
  PROC DATASETS LIB=&LIBRARY MEMTYPE=DATA;
  QUIT;
ODS OUTPUT CLOSE;
%IF &SYSERR NE 0 OR NOT(%SYSFUNC(
  EXIST(WORK.FILELISTER))) %THEN %DO;
  %LET NODATASETS = 1;
  %GOTO NOSASFILES;
%END;
DATA _NULL_;
  SET WORK.FILELISTER(WHERE=(MEMTYPE="DATA"))
  END=LAST;
  CALL SYMPUT("DSN"||COMPRESS(PUT(NUM,12.)),
  MEMNAME);
  IF LAST THEN CALL SYMPUT("NUMFILES",
  COMPRESS(PUT(_N_,12.)));
RUN;
```

The macro proceeds to call the Compress Datasets Macro for each dataset in the library. After the Compress Datasets Macro finishes, global macro variables containing data about each dataset such as number of variables, number of numeric variables, number of observations, dataset compression before the macro, and new compression are reassigned to new macro variables representing each dataset.

```
%DO I = 1 %TO &NUMFILES;
  %PUT &LIBRARY..&&DSN&I--PROCESSING LOOP &I OF
  &NUMFILES;
  %COMPRESS_DATASET(&LIBRARY,&&DSN&I,&DTDNCBFR,
  &DTDNCAFTR);
  %LET NUMVAR&I = &NUMVAR;
  %LET NUMRCVAR&I = &NUMRCVAR;
  %LET NUMOBS&I = &NUMOBS;
  %LET CURRCOMP&I = &CURRCOMP;
  %LET COMPRESSION&I = &COMPRESSION;
%END;
```

The next section of the macro uses ODS and PROC DATASETS to create a second file listing. This dataset is the same as previously created; however, the file size and date modified will be different if the dataset is compressed by the macro. The two file listings are merged together and all the macro variables about each dataset are stored in the dataset ORIGTOPOST, which is a log of the compression activity that took place.

```
ODS OUTPUT DATASETS.MEMBERS=WORK.FILELISTER2;
  PROC DATASETS LIB=&LIBRARY MEMTYPE=DATA;
  QUIT;
ODS OUTPUT CLOSE;
DATA ORIGTOPOST;
  MERGE FILELISTER (IN=A WHERE=(MEMTYPE="DATA")
  RENAME=(FILE_SIZE=ORIG_SIZE))
  FILELISTER2 (IN=B WHERE=(MEMTYPE="DATA")
  RENAME=(FILE_SIZE=POST_SIZE
  LAST_MODIFIED=DT_COMPRESSED));
  BY NUM MEMNAME;
  LENGTH NUM_VARS NUMERIC_VARS 3 NUM_OBS 8
  LIB $7 OLD_COMP NEW_COMP $6;
%DO I = 1 %TO &NUMFILES;
  %IF &I NE 1 %THEN ELSE;
  IF NUM EQ &I THEN DO;
    LIB = "&LIBRARY";
    NUM_VARS = &&NUMVAR&I;
    NUMERIC_VARS = &&NUMRCVAR&I;
    NUM_OBS = &&NUMOBS&I;
    OLD_COMP = "&&CURRCOMP&I";
    NEW_COMP = "&&COMPRESSION&I";
  END;
%END;
IF OLD_COMP EQ NEW_COMP THEN DO;
  DT_COMPRESSED = .;
  NEW_COMP = ' ';
END;
IF SUBSTR(NEW_COMP,1,1) IN ("D","F") THEN
  DT_COMPRESSED = .;
RUN;
```

The rest of the macro simply calculates the space savings and writes this information to the SAS log. At the end of the macro, temporary datasets are deleted. Note that the dataset ORIGTOPOST is only deleted if no datasets exist in the library because it is used in the next macro.

```
DATA _NULL_;
  SET WORK.ORIGTOPOST END=LAST;
  RETAIN ORIGLSIZE ORIGCFSIZE POSTLSIZE;
  ORIGLSIZE + (ORIG_SIZE / 2**20);
  POSTLSIZE + (POST_SIZE / 2**20);
  IF INPUT("&DTSTARTMOD",MMDDYY10.) LT
  DATEPART(LAST_MODIFIED) LT
  MIN(INPUT("&DTENDMOD",MMDDYY10.),99999)
  THEN ORIGCFSIZE + (ORIG_SIZE / 2**20);
  IF LAST THEN DO;
    PCTCFSAVE = ORIGCFSIZE / ORIGLSIZE * 100;
    SAVINGS = ROUND(ORIGLSIZE - POSTLSIZE,
    .001);
    PCTCFSAVE = SAVINGS / ORIGCFSIZE * 100;
    PCTLSAVE = SAVINGS / ORIGLSIZE * 100;
    CALL SYMPUT("ORIGLSIZE",
    COMPRESS(PUT(ORIGLSIZE,12.3)));
    CALL SYMPUT("POSTLSIZE",
    COMPRESS(PUT(POSTLSIZE,12.3)));
    CALL SYMPUT("ORIGCFSIZE",
    COMPRESS(PUT(ORIGCFSIZE,12.3)));
    CALL SYMPUT("PCTLCOMP",
    COMPRESS(PUT(PCTLCOMP,12.1)));
    CALL SYMPUT("SAVINGS",
    COMPRESS(PUT(SAVINGS,12.3)));
    CALL SYMPUT("PCTCFSAVE",
    COMPRESS(PUT(PCTCFSAVE,12.2)));
    CALL SYMPUT("PCTLSAVE",
    COMPRESS(PUT(PCTLSAVE,12.2)));
  END;
RUN;
%IF &SAVINGS NE 0.000 %THEN %DO;
  %PUT #####;
  %PUT ## ORIGINALY, DATASETS IN THE
```

```

" &LIBRARY" LIBRARY WERE &ORIGLSIZE MB.;
%PUT ## FILES WITHIN SPECIFIED DATE RANGE
ARE &PCTLCOMP% OF THIS SPACE, OR
&ORIGCFSAVE MB.;
%PUT ## COMPRESSION SAVED &SAVINGS MB, OR
&PCTCFSAVE%!! OVERALL, THE MACRO;
%PUT ## RESULTED IN A SAVINGS OF
&PCTLSAVE%: " &LIBRARY" LIBRARY IS NOW
&POSTLSIZE MB.;
%PUT #####;
%END;
%ELSE %DO;
%PUT #####;
%PUT ## " &LIBRARY" LIBRARY CONTAINS
&ORIGLSIZE MB OF DATASETS: NO DATASETS
WERE COMPRESSED;
%PUT #####;
%END;
%NOSASFILES:
PROC DATASETS LIB=WORK NOLIST;
DELETE FILELISTER FILELISTER2;
QUIT;
%IF &NODATASETS EQ 1 %THEN %DO;
%IF (%SYSFUNC(EXIST(ORIGTOPOST))) %THEN %DO;
PROC DATASETS LIB=WORK NOLIST;
DELETE ORIGTOPOST;
QUIT;
%END;
%PUT #####;
%PUT ## NO SAS DATASETS EXIST IN THE
" &LIBRARY" LIBRARY.;
%PUT #####;
%END;
%MEND COMPRESS_LIBRARY;

```

### Iterate Library Macro

The Iterate Library Macro uses a directory listing generated by VBA to assign library references to multiple directories and then calls the Compress Library Macro. This macro is designed to be executed through a VBA interface; however, it can also be run directly using SAS. There are four inputs to the macro: OUTLIB, FILELIST, DTDNCBFR, and DTDNCAFTR. OUTLIB and FILELIST reference a library and dataset name that will store a dataset with a log of all the macro's compression activity. The date inputs are optional and are discussed in detail under the Compress Datasets Macro section.

The file reference early in the macro, C:\Folders.txt, is created by the VB interface, and contains a directory listing for all paths below the root directory. An example of the contents of this file is illustrated under The Auto-Compress Macro Interface section, Step 1. The macro uses this file to assign libraries to every directory by reading the file and using CALL SYMPUT to pass the directories to macro language.

```

%MACRO ITERATE_LIB(OUTLIB,FILELIST,DTDNCBFR,
DTDNCAFTR);
PROC PRINTTO LOG="C:/COMPRESSION.TXT" NEW; RUN;
%LOCAL I;
%IF (%SYSFUNC(EXIST(&OUTLIB.&FILELIST)))
%THEN %DO;
PROC DATASETS LIB=&OUTLIB NOLIST;
DELETE &FILELIST;
QUIT;
%END;
FILENAME DIR "C:\FOLDERS.TXT";
DATA WORK.LIBRARY (COMPRESS=CHAR);
LENGTH LIBRARY $150;
INFILE DIR LENGTH=RLNGTH MISSEVER;
INPUT @1 LIBRARY $VARYING150. RLNGTH;
RUN;
DATA _NULL_;

```

```

SET WORK.LIBRARY END=LAST;
CALL SYMPUT("LIB"||COMPRESS(PUT(_N_,12.)),
TRIM(COMPRESS(LIBRARY,'')));
IF LAST THEN CALL SYMPUT("NUMLIB",
COMPRESS(PUT(_N_,12.)));
RUN;

```

The macro loops through each directory, assigns the library reference, ensures that the reference is valid and then calls the Compress Library Macro. The dataset ORIGTOPOST, created by the Compress Library Macro, is used to build the output file containing a summary of all the macro's activity.

```

%DO I = 1 %TO &NUMLIB;
LIBNAME LIB&I "&&LIB&I";
%IF NOT(%SYSFUNC(LIBREF(LIB&I))) %THEN %DO;
%COMPRESS_LIBRARY(LIB&I,&DTDNCBFR,
&DTDNCAFTR);
%IF NOT(%SYSFUNC(EXIST(
&LIBRARY.&FILELIST))) AND
(%SYSFUNC(EXIST(ORIGTOPOST)))
%THEN %DO;
DATA &OUTLIB.&FILELIST;
LENGTH DIRECTORY $150 MEMNAME $35;
SET ORIGTOPOST(DROP=MEMTYPE NUM);
DIRECTORY = "&&LIB&I";
RUN;
%END;
%ELSE %IF (%SYSFUNC(EXIST(ORIGTOPOST)))
%THEN %DO;
DATA &OUTLIB.&FILELIST;
SET &OUTLIB.&FILELIST(IN=A)
ORIGTOPOST(IN=B DROP=MEMTYPE NUM);
IF B THEN DIRECTORY = "&&LIB&I";
RUN;
%END;
LIBNAME LIB&I CLEAR;
%END;

```

The final step of the macro uses a DATA step to calculate space savings and summary statistics. This step is very similar to the DATA \_NULL\_ at the end of the Compress Library Macro. The COMPRESSALL dataset contains the variables that are sent to macro language in the prior macro. The macro exports this dataset and the compression log dataset to Excel where they will be used by the VB interface. The last of the work files are also cleaned up.

```

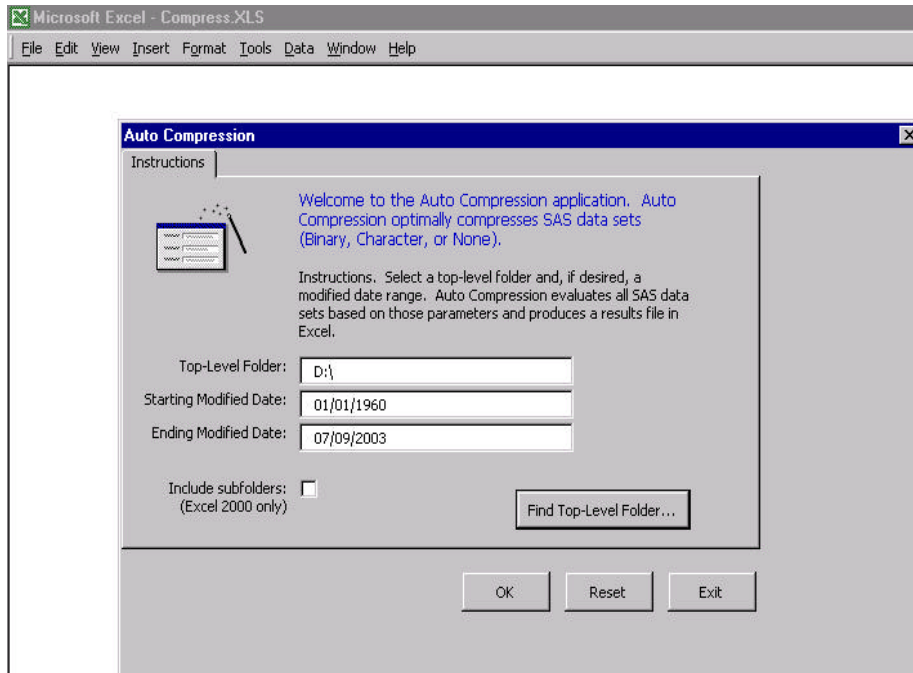
PROC EXPORT DATA= &OUTLIB.&FILELIST
OUTFILE= "C:\AUTOCOMPRESS_RESULTS.XLS"
DBMS=EXCEL REPLACE;
RUN;
PROC EXPORT DATA=COMPRESSALL
OUTFILE= "C:\AUTOCOMPRESS_ALL.XLS"
DBMS=EXCEL REPLACE;
RUN;
PROC DATASETS LIB=WORK NOLIST;
DELETE COMPRESSALL LIBRARY ORIGTOPOST;
QUIT;

PROC PRINTTO; RUN;
%MEND ITERATE_LIB;

```

## THE AUTO-COMPRESS MACRO INTERFACE

A chief reason for building an Excel VBA interface is to supply the Auto-Compress Macro with an essential text file that contains the directory listings that will be evaluated. The interface is relatively transparent. The user remains in Excel while executing SAS. This capability relieves the user of



needing to know the particular macro-language characteristics of the Auto-Compress SAS program, or the SAS language (!).

### Requirements

The simple design and functionality of the interface rely on SAS' technology of Object Linking and Embedding (OLE) automation and Visual Basic for Applications (VBA). OLE automation, available since version 6.12, enables multiple Windows applications to communicate with each other. Visual Basic for Applications (VBA), an instructional language for Word, Excel, PowerPoint, Visio, Access, and Outlook, interacts with the user's inputs and controls SAS.

As displayed above, the Auto-Compress Macro Interface consists of a single user form and a few controls. The user selects a top-level folder where the SAS datasets reside and, optionally, a modification date range. After the user clicks the "OK" button, the VBA code proceeds as follows:

1. Create a text file comprised of multiple directories that will serve as input for the Auto-Compress SAS macro.
2. Create a SAS session, which runs in the background of Excel;
3. Submit the Auto-Compress Macro SAS program;
4. Determine when the SAS program finishes;
5. Close the SAS session;
6. Return the SAS output to Excel. Inform the user of the overall efficiency through a message box, and query the user whether to display a detailed results file in Excel.

### Visual Basic for Applications Statements

The VBA statements of these steps are listed and described below. These are key statements, and, hopefully, sufficiently generic for similar application development.

#### Step 1 - Create a Text File of Multiple Directories

A key component of the Auto-Compress Macro is its ability to iterate through multiple directories and test each SAS dataset for optimal compression. In order to achieve this, the SAS macro requires a text file containing a list of directories. In VBA, the ubiquitous Dir function is unsatisfactory. Fortunately, several

Windows API functions on which the Dir function is based enable recursive file and directory search. In "VBA Developer's Handbook, 2nd Edition," the authors provide Windows API functions that are nearly perfectly suitable for the macro. Only small modifications were necessary: restricting the search to SAS datasets, and altering the code to return the folder only and not the complete path of the file. To initiate Step 1, the VBA code below opens (creates) a text file, calls the API function DhPrintFoundFiles, and passes the top-level directory and file attributes:

```
Open msFolderList For Append As #1 _
Write #1, UserForm1.tbName.Text
If UserForm1.chkSubFolders.Value _
= True Then
Call _
DhPrintFoundFiles _
(UserForm1.tbName.Text, "*.sas7bdat")
End If
Close #1
```

The variable 'msFolderList' can be any string, e.g. 'c:\folders.txt'. The IF statement checks whether subfolders are to be included in the text file, based on the status of the userform's checkbox, and then closes the file. A typical output might look like:

```
"D:\data"
"D:\data\capextend"
"D:\data\cb"
"D:\data\CurrentOV"
"D:\data\finance"
```

The API function DhPrintFoundFiles acts like the Dir function. The text from the userform's textbox serves as the starting directory, and the second parameter "\*"\*.sas7bdat" is the file specification.

#### Step 2 - Create a SAS session

After the user presses the "OK" command button, the following VBA code creates a SAS session:

```
Public Sub OpenSAS ()
' Declare an object variable
```

```

Dim OleSAS As Object
Set OleSAS = CreateObject("SAS.Application")
OleSAS.Visible = False
Application.StatusBar = ""
Exit Sub
End If
Application.StatusBar = "SAS is running."
End Sub

```

The Set statement creates the SAS session. The Visible property set to 'False' places SAS in the background. Populating a message to the StatusBar informs the user that SAS is running.

### Step 3 – Submit the Auto-Compress SAS Macro

The Submit method directly submits SAS code. One elegant way to run a SAS program is incorporating a %INCLUDE statement:

```
OleSAS.Submit ("%include 'c:\
AutoCompress.sas';")
```

VBA can dynamically respond to events during the course of SAS processing. The following example illustrates responses to the event of when SAS has finished.

### Step 4 – Determine when the SAS program has finished

At the end of the Auto-Compress program, SAS creates an Excel output through PROC EXPORT. In VBA, a function tests for this output by using the Wait method and the TimeSerial function. The results of this test confirm whether SAS output is created and also whether SAS is finished.

```

Function WaitSASRunning(filename as String) as Boolean
Dim intHour as Integer
Dim intMinute as Integer
Dim intSec as Integer
Dim Counter as Integer
For Counter = 1 To 20
intHour = Hour(Now)
intMinute = Minute(Now)
intSec = Second(Now) + 15
WaitTime = TimeSerial(intHour, intMinute,
intSec)
Application.Wait WaitTime
` Does the output exist
If Len(Dir$(filename)) then
WaitSASRunning = True
End If
Next Counter
WaitSASRunning = False
End Function

```

The name of our final output is passed as a parameter, 'filename', which must be a fully qualified path and file name such as 'c:\finaloutput.xls'. The Wait method, 'Application.Wait,' pauses until a specified time. In this example, Application.Wait pauses every 15 seconds. The delay is predicated on the specified time in TimeSerial function. After every 15 seconds, the existence of file is tested, through the Len(Dir\$(filename)) statement. If after 20 iterations

(approximately 300 seconds), the test for existence of the file remains unsuccessful, the Function returns false.

### Step 5 – Close the SAS Session

Once the SAS session is instantiated, the SAS object persists until it is removed. The Quit method exits SAS. A good programming practice is to remove the reference to the object from memory when SAS has finished with the Nothing keyword.

```
OleSAS.Quit
Set OleSAS = Nothing
```

### Step 6 -Return SAS output to Excel.

This Sub routine illustrates the WaitSASRunning function. If the result is 'True', that is, if the output exists, then open the workbook. The Open method opens a workbook:

```

Sub ReturnExcelFile
Dim xlOutput as Workbook
If WaitSASRunning = True then
` Open and Save the Excel results file
Set xlOutput = Workbooks.Open(Filename:=
"c:\SASResults.xls", ReadOnly:=True)
xlOutput.Save
End If
End Sub

```

## CONCLUSION

The Auto-Compress Macro can save a significant amount of storage space and time. The automated approach is quick and easy to use; simply point and click. The results of the macro speak for themselves. The first execution of the program on a portion of one of Household International's servers resulted in a savings of 47.18 GB. The macro scanned through 706 SAS datasets and found that 519 of them could be compressed. The SAS datasets originally occupied 117.45 GB of drive space, now the same datasets occupy 70.26 GB of drive space; this is a reduction of 40.2%!

## REFERENCES

VBA Developer's Handbook, Second Edition (2001). Ken Getz and Mike Gilbert. Sybex.

## CONTACT INFORMATION

Adam Bemis  
Household International  
9400 SW Beaverton Hillsdale Hwy, Suite 300  
Beaverton, OR 97005  
503-432-3677  
[Adam.L.Bemis@household.com](mailto:Adam.L.Bemis@household.com)

Scott Hanson  
Household International  
9400 SW Beaverton Hillsdale Hwy, Suite 300  
Beaverton, OR 97005  
503-432-3663  
[Scott.P.Hanson@Household.com](mailto:Scott.P.Hanson@Household.com)