

Breaking Up Is Hard To Do! Understanding the CGM Specification

Michael P. Bramley, Fairfield, OH

ABSTRACT

While the latest versions of The SAS® System have included significant advances in document selection, organization, and publication-ready output, the drivers in the SAS/GRAPH® software have also been updated. One such update is how these drivers implement the Computer Graphics Metafile (CGM) specification. This implementation can adversely affect SAS programs that output multiple graphs into one Graphics Stream File, with the intention of splitting it up on a target system, say for use in word processing systems.

This article includes a brief history of the CGM specification, a gentle introduction to the workings of the CGM specification, and how the implementation of the CGM specification differs between versions of SAS software that result in good old 'gotchas'. With this necessary ground covered, an efficient single-pass algorithm is presented that converts any SAS/GRAPH software created CGM file containing k graphs into k files, where the k^{th} file contains the k^{th} graph. This algorithm is implemented as both a SAS macro and as a C program.

This article has two intended audiences: SAS users who realize benefits from producing multiple graphs in one CGM file for delivery on an alternate platform for inclusion in target documents, and those who must deal with legacy systems (especially SAS 6.12) as a part of their job.

INTRODUCTION

When it comes to producing graphics with SAS software, many users prefer the route of creating Graphic Stream Files (GSF). A GSF is an external file that contains graphic commands that the target software, such as a word processing system, recognises. Two reasons many users choose the CGM format for their GSF are that the hardware fonts provide clear renderings of their subject matter and it permits multiple graphs to be stored in one file.

The ability to store multiple graphs in one external file makes the code for producing graphs of a similar nature almost trivial if the data can be coerced into a form conducive to BY-group processing. Moreover, the CGM format makes file management easier, especially if there are many graphs to collate for a project. Often the process is reduced to ftp'ing the CGM file to the target system where it is broken up by another utility into its individual graphs. The focus of this article is what happens (or doesn't happen) once the CGM file is on that target system...

CGM HISTORY

The American National Standards Institute originally adopted the Computer Graphics Metafile (CGM) specification in 1986 (X3.122-1986) to fulfill the role of a full-featured file format capable of describing complex raster, vector, and hybrid high-resolution graphics compactly. Soon thereafter, it became the franca-lingua for many software packages, including The SAS System. Due to its long-term ease-of-use, reliability, flexibility, and plethora of support, it remains a dominant player in the newly netted world of presentation graphics.

The International Organization for Standards adopted the second version of the CGM specification in 1992 (ISO CGM 8632: 1992), with two amendments. The first amendment, "Conformance, Rules for Profiles, and the Model Profile", provides guidance on writing conforming CGM files and expanding the CGM specification to meet different industrial needs. The second

amendment, entitled "Application Structuring Extensions", provides the ability to tag, reference, and index arbitrary collections of elements with application significance. Modeled after SGML, the new features provide a way to structure objects in a metafile. Application areas targeted for this intelligent graphics capability include electronic review of documents, graphical object databases, and multi-media and hyper-media documents.

The third version of the CGM specification was adopted by the International Organization for Standards in three parts: ISO/IEC 8632-1:1999 (Functional Specification), ISO/IEC 8632-3:1999 (Binary Encoding), and ISO/IEC 8632-4:1999 (Clear Text Encoding). The second part of the specification (Character Encoding) from the 1992 specification has been withdrawn, due to obsolescence. This article focuses on the binary encoded files that meet the ISO/IEC 8632-3 CGM specification.

The most important things to learn from this brief history are that the CGM specification is an old stalwart that has acquired new tricks with time, and that it is a widely accepted standard that has been adopted by many countries, such as Australia, France, Germany, Japan, Korea, and the United Kingdom. The best features of this specification are that it is easy to understand and that all versions are upwards compatible. This means that a version one based CGM file can still be interpreted by a decoder designed for version three CGM files.

THE CGM SPECIFICATION

To conceptualize the structure of a CGM file produced by The SAS System, think of a binary data stream where you see exactly one word (16 bits) at a time. This binary data stream can be divided into a series of logical components that are called elements, as illustrated in Figure 1. An element consists of a one word header set to some predefined value, optionally followed by a variable length parameter list. A parameter list is just CGM terminology to describe the bytes of graphical information that are required to complete an element. For example, the parameter list of a picture element includes the contents of the picture: the axes, the legend, etc. Elements are quite analogous to the tags found in XML or HTML files, with parameter lists standing in for what the tags encapsulate.



Figure 1: Layout of CGM Binary Data Stream

CGM ELEMENTS

CGM elements are the key to understanding the contents of a binary encoded CGM file: they transform a binary data stream into a set of hierarchical objects. There are different classes of elements based on their functionality, of which, this article shall concentrate on the Delimiter Elements that signal the start/end of a logical structure, such as a graph or the CGM file itself. Delimiter Elements are easier to understand as they are always paired together, in the same way as the SAS language requires that each DO statement be paired with an END statement.

Elements in the CGM file may appear in either a short-form or a long-form. The layout of the short-form element is displayed in Figure 2, where the top row of the figure depicts a reference for the 16 bit positions of a word in the binary data stream (bits 15 through 0). Bit 15 is the most significant bit (MSB) and bit 0 is the least significant bit (LSB). The rows under this represent the order of the binary data stream in sequential order (as seen one word at a time).

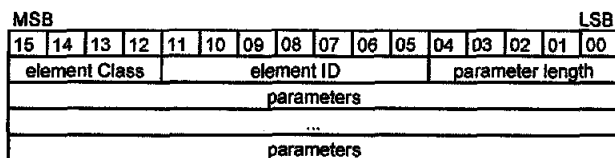


Figure 2: Layout of Short-Form Element

The first word of an element is divided into three components: the four bit element Class in bits 15 through 12, the seven bit element ID in bits 11 through 5, and the five bit parameter list length in bits 4 through 0. If any parameters are included, they follow the first word of the short-form element. The short-form element is only used when the parameter list length is less than 31.

The element Class and ID in the long-form element are the same as described above in the short-form element. The only difference between these two forms is in the parameter list length, which is set to the constant value of 11111 binary (31 decimal) in the long-form element. In this case, the parameter list length is deferred to the second word, where it is interpreted as a signed 16-bit value and should possess a value greater than or equal to 31 (decimal). This is then followed by the actual parameters, as shown in Figure 3.

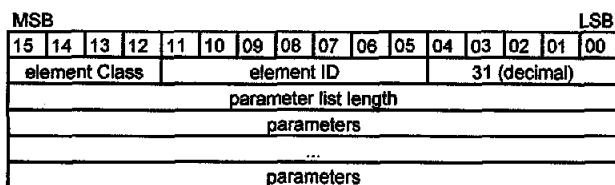


Figure 3: Layout of Long-Form Element

PADDING THE FILE

The CGM specification dictates that elements always begin on a word boundary, which means that the length of each element (including its associated parameter list) must be an even number. This leads to the concept of padding. Padding simply means that a trailing null byte may be appended to an CGM element's parameter list to ensure that subsequent elements always begin on a word boundary. Thus, if an element has an odd number of parameters, then a trailing null byte will be appended to the parameter list (to ensure that the total length is an even number). This trailing null byte is not included in the value of the parameter list length.

While this aspect of the specification may introduce an artificial complexity to software that creates CGM files, it makes the algorithm for decoding CGM files far more simpler. Moreover, it also provides a quick check for CGM files: If the file size is not an even number, then it cannot be a valid CGM file.

BATTILING THE ELEMENTS

In order to break up a CGM file containing multiple graphics, it is necessary to know only four elements—the rest are superfluous when it comes to a CGM file created by The SAS System. For the most part, this means that breaking up a CGM file is simply a matter of scanning for and reacting appropriately to these elements.

All CGM files must start with the Begin Metafile Element, which is depicted in Figure 4 (all numbers in the figures are in decimal). This must be the first word of a CGM file, otherwise the file does not conform to the CGM specification. While this delimiter element has parameters, they are only collected for later use and not checked, as the primary purpose of this article is not to decode the CGM file, but rather break it up.

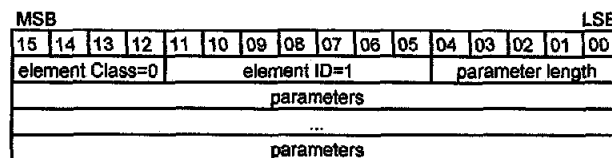


Figure 4: Layout of Begin Metafile Element (Short-Form)

The next element of concern is the Begin Picture Element portrayed in Figure 5. This delimiter element signals the beginning of a picture definition and is repeated to describe multiple graphics within the same CGM file.

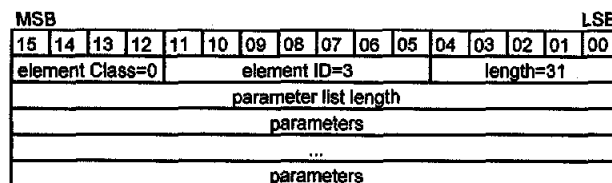


Figure 5: Layout of Begin Picture Element (Long-Form)

Every Begin Picture Element must be paired with an End Picture Element, the form of which is illustrated in Figure 6. This delimiter element signals the end of the current picture definition. It is exactly one word long (it has no parameters), comes in only the short-form, and is a constant value. Matching its occurrence to a previous Begin Picture Element should be incorporated as a test that the file conforms to the CGM specification.

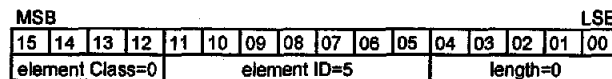


Figure 6: Layout of End Picture Element

The last element to know is, appropriately, called the End Metafile Element. This delimiter element is shown in Figure 7 and indicates the end of the CGM binary data stream. Identification of this element in the binary data stream should terminate any further processing of the CGM file. This element is similar to the End Picture Element in that it is one word in length and a constant value. Moreover, a test should be incorporated to check that the CGM file includes an End Metafile Element.

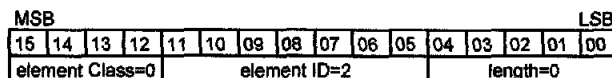


Figure 7: Layout of End Metafile Element

That part of the CGM file from the Begin Metafile Element up to and excluding the first Begin Picture Element is referred to as the CGM header. This information must be written to every CGM file derived from the original CGM binary data stream in order for it to conform to the CGM specification. The remainder of each new CGM file created consists of the original binary data stream from the Begin Picture Element through to the End Picture Element, inclusive, with a trailing End Metafile Element to match up with the Begin Metafile Element included in the CGM header. A file created in this fashion conforms to the CGM specification and can be imported into any software product that supports the CGM specification.

Now that the requisite elements of the CGM specification have been covered, all that remains is to understand how to identify them in the CGM binary data stream. This process is trivial for the End Picture and End Metafile Elements, as their values never change: simply scan the binary data stream for the hexadecimal values of 0x00A0 and 0x0040, respectively. This, however, is not the case for the two remaining elements if the algorithm is to correctly handle CGM files created by any version of The SAS System.

BIT PICKING

To identify the latter two elements in the data stream, it suffices to scan for the relevant element Class and ID. This requires the use of bit-testing as these components are not easily amenable to normal byte testing methods, as the ID does not fall on a byte (or even nibble) boundary.

Bit-testing is a method of testing the individual bit settings of numeric or character data and is implemented in many languages, including SAS. As far as SAS technical support is able to confirm, bit-testing has been a part of SAS since at least version 5, perhaps even earlier.

In general, bit-testing is accomplished by comparing a variable to a bit-mask. The bit-mask is a set of binary digits where a zero tests if the corresponding bit in the variable is off, and a one if the corresponding bit is on. The entire comparison returns true only when all of the bits in the bit-mask match those in the variable. In ASCII-land, for example, if the character variable `Initial` equals the value 'C', then comparing it to the bit-mask "0100 0011"b will return true [1]. Note that in this article a space is added every four bits to improve legibility, and may be specified this way in SAS programs.

If the bit-mask is partial, that is, ignores specific bits, then a common practice is to turn those bits off in the variable and set the corresponding bits in the bit-mask to zero. This is far too much work and SAS provides a simpler solution which permits the use of the period in the bit-mask to indicate "ignore", that is, the corresponding bit in the variable is not tested. Assuming the former variable `Initial` contains only values of the Roman alphabet, one can test if this variable is uppercase (in ASCII) by comparing it to the bit-mask of ".0."b, which returns true only when the third bit in `Initial` is zero (indicating an uppercase letter).

A LITTLE BIT AT A TIME

The recognition of the Begin Metafile and Begin Picture Elements can now be achieved with two simple comparisons by utilizing bit-testing and the fact that the only variable component in the two elements is the parameter list length.

Referring to Figure 4, the bits containing the element Class and ID for the Begin Metafile Element are 15 through 5 inclusive, with values of zero and one, respectively. Only bits 4 through 0 vary, depending upon whether the element is in short-form or long-form, and can be safely ignored for the time being. From most to least-significant bit, the bit-mask will be made up of four zeros for the Class, six zeros and a one for the ID, and five periods for the parameter length, which is ignored. Thus, the bit-mask to test for this element shall be: "0000 0000 001."b. Similarly, the bit-mask for the Begin Picture Element can be created by applying the same method to Figure 5, resulting in "0000 0000 011."b.

If the intent is to interpret and/or display the individual components of a CGM file, then upon determining one of these elements, it suffices to extract the parameter list length from bits 5 through 0. If this value is less than 31 (decimal), then the element is in short-form and the value is the actual parameter list length. Otherwise, this is a long-form element and the next word must be obtained to determine the parameter list length.

NEW VERSIONS YIELD NEW PROBLEMS

The older SAS/GRAPH CGM drivers appear to produce only CGM files with long-form elements. This changed with the introduction of version 8 of The SAS System, which produces CGM files with short-form and/or long-form elements. In particular, the form of the Begin Picture Element changed. Thus, users who wrote programs that scanned a multi-picture CGM file for long-form Begin Picture Elements with a constant hexadecimal value of 0x007F found that their programs failed to function with the CGM files produced by the latest releases of The SAS System.

This is not an error in The SAS System, but rather the SAS developers' decision to utilise more of the CGM specification. The result is that code, such as the CGMSplit macro [2], is rendered useless when it comes to CGM files created by the latest versions of The SAS System.

PUTTING IT TOGETHER TO BREAK IT APART

The simplified BREAKUP algorithm to break up a CGM file is quite straightforward, given the preceding details. Note that the algorithm below assumes that all administrative details have been performed (initialization of variables, opening and closing files, etc.), and that the current word from the binary data stream is denoted by CW.

```
Step 0.  Read first CW.
        IF CW = Begin Metafile Element
        THEN ProcessHeader
        ELSE Exit with Error

Step 1.  LOOP Until( CW = End Metafile Element )
        Read CW.
        IF CW = Begin Picture Element
        THEN ProcessPicture
        ELSE ProcessHeader
```

Step 0 ensures that the first word of the CGM file is, in fact, a file that starts with the requisite Begin Metafile Element. If this test fails then the algorithm exits, otherwise it adds this word to the CGM header.

Step 1 is a simple loop that continually reads one word at a time from the CGM file until it encounters the End Metafile Element. In the loop, one of two branches are made: either process the current word as the beginning of a picture, or as a component of the CGM header.

The ProcessHeader routine is a trivial block that stores words of the binary data stream in memory for later when they have to be written to a new CGM file. The ProcessPicture algorithm presented below is a simple I/O copy routine with extra steps to write the CGM header and End Metafile Element. Note that each time Step 0 is executed, the value of the current word is that of the Begin Picture Element that triggered the call to this algorithm.

```
Step 0.  Output CGM Header.
        Output CW.

Step 1.  LOOP Until( CW = End Picture Element )
        Read CW.
        Output CW.

Step 2.  Output End Metafile Element.
```

Thus, the basic BREAKUP algorithm is reduced to two loops: the outer loop searching for the trigger that signals the beginning of a picture and the inner loop which creates a new CGM file by writing the CGM header and copying the picture from the original CGM binary data stream to the new CGM file. Moreover, this algorithm requires only a single pass, not multiple passes, of the original CGM file to process it.

ON THE DESKTOP

Implementing the BREAKUP algorithm as a SAS/MACRO® is quite straightforward. The macro names the new CGM file by appending a counter representing the graph's relative position in the file to the original CGM filename, for example, CGMFILE-00001.cgm. The macro also accepts parameters that allow the user to dictate the location of the new CGM files as a) the same path as the original CGM file, or b) a different user-specified path. A further parameter places the new CGM files into a sub-path

(named after the CGM filename). The inclusion of a user-specified path is meant for those projects which have a common path for graphics files, for example, G:/Data/Team-Project/Phase-III/Graphs/.

Before the main macro can be discussed, two macro functions must be introduced that are used by the BreakUp macro. The CheckPath() macro function returns one if the path exists or was successfully created, and zero if the path does not exist or could not be created. The code for this macro function follows:

```
00 %Macro CheckPath(Path, Create) ;
01   %Local FileRef DID RC ;
02   %Let RC = 0 ;

03   %If Not %SysFunc(FileName(FileRef, &Path))
    %Then %Do ;
04     %Let DID = %SysFunc(DOpen(&FileRef)) ;
05     %If Not &DID %Then %Do ;
06       %If &Create GT 0 %Then %Do ;
07         %SysExec Mkdir "&Path" ;
08         %Let RC=%Eval(Not &SysRC) ;
09         %Put CheckPath: Creating &Path.. ;
10       %End ;
11     %End ;
12   %Else %Do ;
13     %Let RC=1 ;
14     %Let DID=%SysFunc(DClose(&DID)) ;
15   %End ;

16   %Let DID=%SysFunc(FileName(FileRef)) ;
17 %End ;
18 &RC
19 %Mend CheckPath ;
```

Lines 01 and 02 declare the local macro variables used by the macro function, and set the return code (RC) pessimistically. Line 03 tests if a temporary file reference can be setup for the path. In particular, The SAS System fills the macro variable FileRef with a unique file reference of the form #LNkkkkk, where k is zero-filled counter that the SAS software maintains internally.

If this succeeds, the macro function attempts to open the path at line 04 with the SAS I/O function DOPEN. If the open fails, then the macro function attempts to create the path if that option was chosen (lines 05 through 11). If the path does exist, then lines 13 through 14 are executed, which set the return code and close the path with the DCLOSE function. The temporary file reference is removed from the SAS metadata and the return code returned to the invoking procedure. Note that the SAS I/O functions DOPEN and DCLOSE were added to SAS9.

The CheckFile() macro function returns one if the external file exists and zero otherwise. The code for this macro function follows:

```
00 %Macro CheckFile(Path, File) ;
01   %Local FileRef FID RC ;
02   %Let RC = 0 ;

03   %If Not
    %SysFunc(FileName(FileRef, &Path/&File))
    %Then %Do ;
04     %Let FID = %SysFunc(FOpen(&FileRef)) ;
05     %If &FID %Then %Do ;
06       %Let RC=1 ;
07       %Let FID=%SysFunc(FClose(&FID)) ;
08     %End ;
09   %Let FID=%SysFunc(FileName(FileRef)) ;
10 %End ;
```

```
11   &RC
12 %Mend CheckFile ;
```

Lines 01 and 02 declares the local macro variables the macro function uses, including setting the return code (RC) to a pessimistic failure. Line 03 tests if a temporary file reference can be setup for the external filename. If this succeeds, the macro function attempts to open the external file for input at line 04 with the SAS I/O function FOPEN. If the open succeeds, then the macro function sets the return code and closes the connection to the external file with the FCLOSE function (lines 05 through 08). The temporary file reference is then removed from the SAS metadata and the return code returned to the invoking procedure.

The following code presents the BreakUp macro, which breaks up binary encoded CGM files containing k pictures into k separate CGM files, where the kth CGM file created contains the kth picture. The author apologies in advance for the indentation challenges made possible by this multi-column format. As per the previous discussion, the four parameters to this macro are:

CGMPath = pathname of the CGM file,
CGMFile = existing CGM filename (minus the extension),
SubDir = a sub-path indicator (activated by any non-blank value),
AltPath = existing alternate path for the output CGM files.

```
00 %Macro BreakUp(
    CGMPath, /* CGM filename path */
    CGMFile, /* CGM filename (no ext) */
    SubDir,  /* Sub-path mode ? */
    AltPath, /* Alternate path */
) ;

/* Initialization and SAS settings... */

01 %Local CGMRef RC XWait XSync
    MPrint MLogic SymbolGen ;

02 %Let Notes = %SysFunc(GetOption(Notes)) ;
03 %Let XWait = %SysFunc(GetOption(XWait)) ;
04 %Let XSync = %SysFunc(GetOption(XSync)) ;
05 %Let MPrint = %SysFunc(GetOption(MPrint)) ;
06 %Let MLogic = %SysFunc(GetOption(MLogic)) ;
07 %Let SymbolGen
    =%SysFunc(GetOption(SymbolGen));

08 Options NoNotes NoMPrint NoMLogic
    NoSymbolGen NoXWait XSync ;

09 %Let RC = %SysFunc(FileName(CGMRef, .)) ;

/* Check that the path and CGM file exist. */

10 %If Not %CheckPath( &CGMPath ) %Then
11   %Put ERROR: Unable to access path of CGM
    file: &CGMPath.. ;
12 %Else
13 %If Not %CheckFile( &CGMPath/&CGMFile..cgm )
    %Then
14   %Put ERROR: Unable to access CGM file:
    &CGMPath/&CGMFile..cgm. ;
15 %Else %Do ;

/* If no Alt. path, create CGM files in the
same path as the original CGM. Check if
sub-path mode has been activated. */

16 %If "&AltPath" EQ "" %Then
17   %Let AltPath = &CGMPath ;
18 %If "&SubDir" NE "" %Then
```

```

19 %Let AltPath = &AltPath/&CGMFile ;

/* Proceed if path for new CGM files exists. */

20 %If %CheckPath( &AltPath, 1 ) %Then %Do ;
21   Data _Null_ ;
22   Length CWord $2 NewFile $1024 ;
23   Array Header{2048} $2. ;
24   HdrIdx = 1 ;

25   Put "Breaking Up &CGMPath/
      &CGMFile..cgm ..." ;

26   InFile "&CGMPath/&CGMFile..cgm"
      RecFM = N LRecl = 256 Eof = CGMError ;

27   Input CWord $Char2. @@ ;
28   If CWord EQ "0000 0000 001. ...."b Then
29     Header{HdrIdx} = CWord ;
30   Else Do ;
31     ERROR "ERROR: File does not start
          with Begin Metafile Element.";
32     Stop ;
33   End ;

34   Do Until( CWord EQ '0040'x ) ;
35     Input CWord $Char2. @@ ;

36     If CWord EQ "0000 0000 011. ...."b
37       Then
38         Link ProcessPicture ;
39       Else Do ;
40         HdrIdx + 1 ;
41         If HdrIdx LE Dim(Header) Then
42           Header{HdrIdx} = CWord ;
43         Else Do ;
44           ERROR "ERROR: Header maximum
          exceeded. Probable non-
          conforming CGM file." ;
45         Stop ;
46       End ;
47     End ;

48   PutLog Count "CGM file(s) created." ;
49   Stop ;

50 CGMError:
51   ERROR 'ERROR: End of File reached
      before End Metafile/End Picture
      Element' /
      'ERROR: This file does not
      conform to the CGM
      specification. Do not use any
      CGM files created from it.' ;
52   Stop ;

53 ProcessPicture:
54   Count + 1 ;
55   NewFile = "&AltPath/&CGMFile-" ||
      Put(Count, Z5.) || ".cgm" ;

56   PutLog "Creating " NewFile " ..." ;

57   File &CGMRef FileVar = NewFile
      RecFM = N LRecl = 256 ;

58   Do I=1 To HdrIdx ;
59     Put Header{I} $Char2. @@ ;
60   End ;
61   Put CWord $Char2. @@ ;

```

```

62   Do Until( CWord EQ '00A0'x ) ;
63     Input CWord $Char2. @@ ;
64     Put CWord $Char2. @@ ;
65   End ;

66   Put '0040'x @@ ;
67   Return ;
68   Run ;
69 %End ;
70 %End ;

/* Clean up ... */

71 %Let RC = %SysFunc(Filename(CGMRef)) ;
72 Options &XWait &XSync &Notes
      &MPrint &MLogic &SymbolGen ;
73 %Mend BreakUp ;

```

Lines 01 through 09 perform the declaration and initialization of macro variables used in the BreakUp macro. The bulk of this code obtains and stores the current system options via the GetOption() function so that the macro can set these options to another value before proceeding. Note that these options shall be restored prior to exiting the macro. The last item of business for this block of code is to establish a temporary file reference to the current path (which should always be possible) for later use by the dataset.

Lines 10 through 14 check that the specified path and filename of the CGM file to be broken up exist using the previously discussed macro functions CheckPath() and CheckFile(). Lines 16 through 19 establish the value of the destination path for the new CGM files that are to be created in the macro variable AltPath. Note the interaction between the SubDir, CGMPath, and AltPath macro parameters. That is, the SubDir option is applied regardless of whether or not the user has specified an alternate path.

Line 20 compiles and executes the dataset that actually breaks up the CGM file (lines 21 through 68) only if the destination path for the CGM files exists.

Lines 22 through 26 of this dataset define and initialise variables and establish a logical connection to the specified CGM file to be broken up. The INFILE options of RECFM=N and LRECL=256 ensure that the dataset reads the CGM file as a binary data stream with a logical record length of 256 bytes (the default on most platforms). The EOF=CGMError option tells SAS that any attempt to read beyond the end of file should result in a transfer of control to the code block labeled CGMError. This event should not occur with files that conform to the CGM specification.

Lines 27 through 33 test that the input CGM file conforms to the specification by checking that the first word in the file is a Begin Metafile Element. If so, it simply adds the element to the array labeled Header, else it writes an error message to the log and stops processing.

Lines 34 through 47 implement the outer loop of the BREAKUP algorithm that scans the CGM file for occurrences of the Begin Picture Element. If found, it calls the ProcessPicture routine, else it assumes that the current word is a component of the CGM header and saves it in the array labeled Header. Lines 40 through 45 prevent the code from reading more than 2048 words before determining that the CGM file does not conform to the CGM specification. It has been this author's experience that this value is more than sufficient for CGM files produced by The SAS System.

Lines 50 through 52 are executed only if the dataset attempts to read beyond the end of file, indicating that the input CGM file does not possess either an End Picture Element or an End MetaFile Element, and thus, does not conform to the CGM

specification. An error message is output to the log and the dataset stops. Note that any CGM files created as a result of this should not be used.

Lines 53 through 67 implement the inner loop of the BREAKUP algorithm that creates the individual CGM files by first writing the CGM header (lines 58-60), then the Begin Picture Element (line 61), the parameters and End Picture Element (lines 62-65), and finally the End Metafile Element (line 66). Lines 54 through 57 control the creation of the new CGM files. Note the use of the RECFM=N and LRECL=256 options in the FILE statement to create an output binary data stream. Lines 54 and 55 create a unique filename during dataset *execution* (not compilation), which is stored in the variable `newFile`. This variable is linked to the FILE statement by the FILEVAR option, which effectively permits the dataset to create any number of external files and to defer the naming of said files until execution. This latter point is the most significant: by design, the BREAKUP algorithm cannot know in advance how many pictures are contained in the CGM file, nor can it know in advance the names of the CGM files to be created, without making the resulting code more complex than necessary.

Lines 71 and 72 reverse the changes this macro may have made to The SAS System environment by deleting the temporary file reference and restoring system options to their previous values.

For those users who wish to implement this macro on a SAS V8 system, the PUTLOG statements at lines 48 and 56 must be changed to either ERROR statements or to the FILE LOG; PUT statement groups in order to ensure that the message is delivered to the log and not to the current CGM file being created. The ERROR statement writes a message to the log and sets the automatic dataset variable `_ERROR_` equal to one. However, the DOPEN and DCLOSE SAS I/O functions are new to SAS9, and thus, the invocation of the CheckPath() macro function must either be deleted (and assume that users always correctly specify the paths) or rewritten using only the Filename function.

C:> COMMAND NOT FOUND: SAS

What if your organization does not purchase a license of The SAS System for the target platform that the CGM files are to reside on? This author has quite conveniently addressed this contingency by writing a Borland/GNU compliant C program that implements the BREAKUP algorithm. Both the source and executable are available from the author at the email address below.

CONCLUSION

This article introduces the CGM specification and its history in an abbreviated format that is relevant to SAS users. It also explains the differences that exist in CGM files created by recent versions of The SAS System. Moreover, this article demonstrates an efficient one-pass algorithm to break up binary encoded CGM files created by The SAS System using either a SAS/MACRO or a C program, regardless of which version of SAS produced it.

It is the author's belief that the algorithm and utilities introduced in this article will be useful to SAS users as long as they maintain or update legacy code or write code that creates files containing multiple pictures using the CGM specification to produce high-resolution graphics.

REFERENCES

1. The 'b' after a quoted number of zeros and ones indicates to the SAS compiler that the string is to be converted to a binary number.
2. SAS Observations, Fourth Quarter, 1996.
<http://ftp.sas.com/techsup/download/observations/4q96/kossler/cgmsplit.html>

3. Copies of the 68 page ISO/IEC 8632-3:1999 binary encoding CGM specification (English PDF) may be obtained from the ISO for 156,00CHF (approx. \$121USD) by searching for the ISO publication code of 8632 at the ISO Store's website:

<http://www.iso.org/iso/en/prods-services/ISOstore/store.html>

Moreover, free (and somewhat out-of-date) copies of the CGM specification may be downloaded by googling the phrase "CGM" or "Computer Graphics Metafile".

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Please feel free to contact the author at:

Michael Bramley
223 Martha Lane
Fairfield, OH 45014
W: 513-622-1489
M: 513-315-3756
mbramley@cinci.rr.com

Cha Gheill !!