

## A Program to Split Computer Graphics Metafiles into Individual Graphs<sup>1</sup>

Many SAS<sup>2</sup> users want to include output from SAS/GRAPH procedures in documents that they have created with word processors such as Microsoft Word. The most convenient way to do this is to create a Computer Graphics Metafile (CGM) with one of the SAS/GRAPH CGM device drivers designed for specific word processors (the CGMMW6C driver for Microsoft Word 6.0, for example). The CGM file can then be included in the document that you have created with the word processor.

This process works easily when you are creating a single graph. However, if you use BY-group processing with a SAS/GRAPH procedure to create multiple graphs, or the procedure itself produces multiple graphs, all of the graphs are placed in a single CGM file. This situation can create problems, because most word processors (including Microsoft Word) can only process one graph from a CGM file. (If a file with multiple graphs is included in a Microsoft Word document, only the first graph is processed).

To circumvent this limitation, we have developed a SAS program that reads a single physical CGM file containing multiple graphs and splits it into a separate file for each graph. This program is not only useful in situations where you are doing BY-group processing; it also lets you append graphs from multiple procedures to a single file (using GSFMODE=APPEND) and split them out later.

If you create a CGM file containing two graphs using BY processing or running a single procedure to produce two graphs, the structure of the physical file is as follows:

```

"Beginning of Metafile" flag      (Picture 1)
Metafile Header
"Beginning of Picture" flag
Picture
"End of Picture" flag
"Beginning of Picture" flag      (Picture 2)
Picture
"End of Picture" flag
"End of Metafile" flag

```

In this case, there is a single „Beginning of Metafile“ flag, metafile header, and „End of Metafile“ flag in the physical file.

If you create a CGM file by running two SAS/GRAPH procedures and specifying GSFMODE=APPEND, the structure of the physical file is as follows:

```

"Beginning of Metafile" flag      (Picture 1)
Metafile Header
"Beginning of Picture" flag
Picture
"End of Picture" flag
"End of Metafile" flag
"Beginning of Metafile" flag      (Picture 2)
Metafile Header
"Beginning of Picture" flag
Picture
"End of Picture" flag
"End of Metafile" flag

```

In this case, each SAS/GRAPH procedure produces „Beginning of Metafile“ and „End of Metafile“ flags as well as a metafile header. The SAS/GRAPH CGM drivers treat each picture as a separate metafile, even though multiple pictures may be appended in a single physical file.

The program uses the following steps to split the CGM files:

1. Opens the CGM file, determines the name of the file, and the record length of the file. Then it creates a directory with the same name as the file, into which the individual pictures will be placed.
2. Reads the CGM file, and creates two SAS data sets from it. One data set contains the metafile data itself, and the second contains the beginning and ending locations of each header and picture within the CGM file.
3. Prints out the beginning and ending header and picture locations determined in the second step. This step is optional, and can be used for debugging purposes.
4. Creates an individual CGM file for each picture by splitting the contents of the first SAS data set created in Step 2.

<sup>1</sup> Reprinted with permission from Observations<sup>®</sup> Vol. 6 No. 1 © 1996 SAS Institute Inc.

<sup>2</sup> SAS and SAS/GRAPH are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries.

<sup>®</sup> indicates USA registration.

Other branch and product names are registered trademarks or trademarks of their respective companies.

```

%let urcgm=cgm-file-name; /* CGM input file for */
/* splitting. */
select (status); /* STATUS variable */
/* indicates what we are */
/* looking for next. */

filename urcgm "&urcgm";
options noxwait;

/* Step 1: Process the CGM input filename, and */
/* determine record length of CGM file. */
/* Then create a directory with the same name */
/* as the CGM file. This directory will contain */
/* the individual pictures. */

data _null_;
  call symput('dir',reverse(scan(reverse(trim(symget
    ('urcgm')),2,'.'))));
  infile urcgm recfm=n length=lrecl;
  input h $char1.;
  call symput('lrecl',left(put(lrecl,5.)));
  stop;
run;
%put DIR=&dir LRECL=&lrecl; /* Echo file name */
/* and lrecl. */
dm 'x "mkdir &dir";' /* Create directory with same */
/* name as CGM file. */

/* The following statement is optional to */
/* delete all files PICT*.CGM from the */
/* directory. */
dm 'x "del &dir\pict*.cgm > nul";';

/* Step2 : Store contents of metafile in a SAS data */
/* set (HEX) and store begin and end pointer */
/* for each header and picture in another SAS */
/* data set (LOC). LOC contains one record */
/* for each picture in the file. Note that the */
/* structure of the CGM file will depend on */
/* whether it was created by a single */
/* SAS/GRAPH procedure or by multiple */
/* procedures using */
/* GOPTIONS GSFMODE=APPEND. */

data hex (keep=hex1-hex&lrecl)
  loc (keep=head1 head2 pict1 pict2);
  array hex(*) $ 1 hex1-hex&lrecl; /* Contents of */
/* CGM record. */

/* The following arrays contain the hex */
/* codes that denote the beginning and end */
/* of various components of the metafile. */
array bom(*) $ 1 bom1-bom2 ('00'x '3c'x); /* Codes */
/* for start of metafile. */
array bop(*) $ 1 bop1-bop2 ('00'x '7F'x); /* Codes */
/* for start of picture. */
array eop(*) $ 1 eop1-eop3 ('00'x 'A0'x '00'x);
/* Codes for end of picture. */
array eom(*) $ 1 eom1-eom2 ('7F'x '40'x);
/* Codes for end of metafile. */

retain head1 head2 pict1 pict2;
retain status (0);
retain k (1);
length c $1;
infile urcgm recfm=n eof=eof;
do i=1 to &lrecl;
  input c $char1.; /* Read next byte of metafile. */
  hex(i)=c; /* Store this byte in an array. */
  nc+1;
  if c='20'x then continue; /* Skip blanks. */

  when (0) /* Look for code */
/* indicating beginning */
/* of metafile. */
  do;
  if c=bom(k) then k+1;
  if k=2 and '30'x<=c<='3F'x then k+1;
  else k=(c=bom1)+1;
  if k=3 then do; /* Metafile header has */
/* been found. */
    status=1;
    k=1;
    head1=nc-1; /* Location of */
/* beginning of header. */
  end;

  when (1) /* Look for code */
/* indicating beginning */
/* of picture. */
  do;
  if c=bop(k) then k+1;
  else k=(c=bop1)+1;
  if k=3 then do; /* Beginning of picture */
/* has been found. */
    status=2;
    k=1;
    head2=nc-2; /* Location of */
/* end of header. */
    pict1=nc-1; /* Location of */
/* beginning of picture. */
  end;

  when (2) /* Look for code */
/* indicating end of */
/* picture. */
  do;
  if c=eop(k) then k+1;
  else k=(c=eop1)+1;
  if k=4 then do; /* End of picture */
/* has been found. */
    status=3;
    k=1;
  end;

  when (3) /* Now check to see if we are at end*/
/* or if there is another picture. */
/* If the next character is a '7F'x, */
/* then another picture follows. */
/* If the next character is a '40'x, */
/* we have encountered an */
/* "end of metafile" flag. */
  do;
  if c=eom1 then do; /* If we find another */
/* picture, prepare */
/* to extract it. */
    status=2; /* Location of end */
/* of previous picture. */
    pict2=nc-2; /* Output info about */
/* previous picture to */
/* LOC. */
    pict1=nc-1; /* Location of */
/* beginning of next */
/* picture. */
  end;
  if c=eom2 then do; /* If we hit end of */

```

```

                /* metafile flag ... */
                status=0; /* Prepare to read */
                /* next metafile header. */
                pict2=nc-2; /* Location of end of
                /* previous picture. */
                output loc; /* Output info about
                /* previous picture
                /* to LOC. */
                end;
                if status=3 then do; /* If we are not at end
                /* of picture, read next
                /* character. */
                        status=2;
                        k=(c=eop1)+1;
                end;
                end;
                end;
eof: output hex; /* At end of processing metafile
                /* record, output whole record to HEX. */
run;

/* Step 3: Print the beginning and end location of each
/* header and picture in the physical file, and
/* create a macro variable N containing
/* the number of pictures.

title "&dir";
footnote;
proc print data=loc;
data _null_;
    set loc nobs=n;
    call symput('n',left(put(n,5.)));
    stop;
run;
%put N=&n (number of pictures containing in urcgm);

/* Step 4: Create a single CGM output file per picture
/* by direct access to WORK.LOC (beginning
/* and end locations of the actual picture) and
/* WORK.HEX (CGM input file stored as
/* a SAS data set).

%macro pict(n);
%local i;
%do i=1 %to &n; /* Go through this loop
                /* once for each picture. */
                filename cgm "&dir\pict&i..cgm"; /* Create a
                /* separate file for each picture. Files will
                /* be named PICT1.CGM, PICT2.CGM and
                /* so on. In this step, we move along the
                /* input buffer (array HEX) and write the
                /* information to an output buffer (array PIC),
                /* resetting the start point with each
                /* new header.
                data _null_;
                array hex(*) $ 1 hex1-hex&lrecl; /* Input record.
                array pic(*) $ 1 pic1-pic&lrecl; /* Output record.
                array eof(*) $ 1 eof1-eof2 ('00'x '40'x);
                retain k (0); /* Position of last
                /* character stored in
                /* the output record.
                file cgm recfm=n; /* Open CGM output file
                p=&i; /* Get picture number.
                set loc point=p; /* Read observation from LOC
                /* that corresponds to picture
                /* number. Each observation
                /* contains values for the
                /* beginning(HEAD1) and end*/
                /* (HEAD2) of header and
                /* beginning (PICT1) and end */
                /* (PICT2) of picture body.
                h1=ceil(head1/&lrecl); /* First observation of HEX
                /* containing a
                /* part of the header.
                h2=ceil(head2/&lrecl); /* Last observation of HEX
                /* containing a part of the
                /* header.
                c1=head1-(h1-1)*&lrecl; /* First character of the
                /* header in the observation
                /* that corresponds to H1.
                c2=head2-(h2-1)*&lrecl; /* Last character of the
                /* header in the observation
                /* that corresponds to H2.
                do i=h1 to h2; /* Process each record
                /* containing a part of
                /* the header.
                        set hex point=i; /* Read observation of HEX.
                        if i=h1 then j1=c1; /* J1 contains the starting
                        /* point for processing the
                        /* actual header record
                        /* byte by byte.
                        /* If the first header record
                        /* is being processed, the
                        /* starting point is C1.
                                else j1=1; /* For all subsequent header
                                /* records, the starting point
                                /* is constant 1.
                                        if i=h2 then j2=c2; /* J2 contains the end point
                                        /* for processing the actual
                                        /* header record byte by byte.
                                        /* If the last header record
                                        /* is being processed,
                                        /* the end point is C2.
                                                else j2=&lrecl; /* For all previous header
                                                /* records, the end point
                                                /* is constant LRECL.
                                                        do j=j1 to j2; /* Process each byte of
                                                        /* header record.
                                                                if k=&lrecl then do; /* If output record PIC
                                                                /* is filled,
                                                                /* reset next output record
                                                                /* position and write header
                                                                /* data in chunks of bytes
                                                                /* equal to LRECL of input file.
                                                                        put (pic1-pic&lrecl) ($char1.);
                                                                end;
                                                                k+1; /* Copy the actual header
                                                                /* byte to the next
                                                                /* output record byte.
                                                                pic(k)=hex(j);
                                                                end;
                                                                end;
                                                                /* Begin processing of the
                                                                /* picture body.
                                                                p1=ceil(pict1/&lrecl); /* First observation of HEX
                                                                /* containing a part of the
                                                                /* picture body.
                                                                p2=ceil(pict2/&lrecl); /* Last observation of
                                                                /* HEX containing a part
                                                                /* of the picture body.
                                                                c1=pict1-(p1-1)*&lrecl; /* First character of the
                                                                /* picture body in the
                                                                /* observation that
                                                                /* corresponds to P1.

```

```

c2=pict2-(p2-1)*&lrecl; /* Last character of the */
/* picture body in the */
/* observation that */
/* corresponds to P2. */
do i=p1 to p2; /* Process each record */
/* containing a part of */
/* the picture body. */
set hex point=i; /* Read observation of HEX. */
if i=p1 then j1=c1; /* J1 contains the starting */
/* point for processing the */
/* actual picture body record */
/* byte by byte. */
/* If the first picture body */
/* record is being processed, */
/* the starting point is C1. */
else j1=1; /* For all subsequent picture */
/* body records, the starting */
/* point is constant 1. */
if i=p2 then j2=c2; /* J2 contains the end point */
/* for processing the actual */
/* picture body record. */
/* If the last picture body */
/* record is being processed, */
/* the end point is C2. */
else j2=&lrecl; /* For all previous picture */
/* body records, the end */
/* point is constant LRECL. */

do j=j1 to j2; /* Process each byte of */
/* picture body record. */
if k=&lrecl then do; /* If output record PIC */
/* is filled, reset next output */
k=0; /* record position and write */
/* picture body data in */
/* chunks of bytes */
/* equal to LRECL of input file. */
put (pic1-pic&lrecl) ($char1.);
end;
k+1; /* Copy the actual picture */
/* body byte to the */
/* next output record byte. */
end;
end; /* Close the output metafile. */
do j=1 to 2; /* Process each eof-byte */
/* (00'x and '40'x). */
if k=&lrecl then do; /* If output record PIC */
/* is filled, */
k=0; /* reset next output record */
/* position and write picture */
/* body data in chunks of */
/* bytes equal to LRECL */
/* of input file. */
put (pic1-pic&lrecl) ($char1.);
end;
k+1; /* Fill the actual end-of- */
/* file-byte to the */
pic(k)=eof(j); /* next output record byte. */
end;
do j=k+1 to &lrecl; /* Clean the remaining */
/* characters of the last */
pic(j)='20'x; /* output record with blanks. */
end;
/* Write the last record of */
/* output CGM. */
put (pic1-pic&lrecl) ($char1.);
stop;
run;
%end;
%mend pict;

%pict(&n)

/* clear used filerefs */
filename urcgm clear;
filename cgm clear;

```

## Program Availability<sup>2</sup>

The program CGMSPLIT.SAS is available online through Anonymous FTP and SAS Institute's Bulletin Board System (SIBBS). To access these services, see the instructions inside the front cover of the journal.

## Conclusion

This program can be used to split files created by any SAS/GRAPH CGM driver that generates a binary CGM file. To look at the structure of a CGM file, you can create a graphics stream file using the CGMCLEAR driver. The CGMCLEAR driver writes out the CGM codes in a readable, English format.

If you have CGM files created with BY-processing, or by procedures that generate multiple graphs, an alternative to this program is to use the GIMPORT procedure to read a CGM file with multiple pictures and split them into individual GRSEG entries in a SAS catalog. You can then use PROC GREPLAY to write out each picture to an individual file. Note, however,

<sup>2</sup> <http://www.sas.com/techsup/download/observations/4q96/kossler/cgmsplit.sas>

that PROC GIMPORT cannot be used to split files created by multiple SAS/GRAPH procedures with GSFMODE=APPEND.

Although this program is used to split a CGM file created by SAS/GRAPH software, the program (and PROC GIMPORT) can also be used to split binary CGM files created by other graphics applications as well.

Wolfgang Kössler, Wolf F. Lesener, Mike Kalt

The developers of the code may be reached at the following addresses:

Dr. Wolfgang Kössler  
Humboldt-Universität zu Berlin  
Institut für Informatik  
Lindenstr. 54a  
10099 Berlin  
Koessler@informatik.hu-berlin.de

Wolf F. Lesener  
Humboldt-Universität zu Berlin  
Rechenzentrum  
Unter den Linden 6  
10099 Berlin  
wlesener@rz.hu-berlin.de